# Computer Architecture

Daniel Page

Department of Computer Science,
University Of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB. UK.
⟨csdsp@bristol.ac.uk⟩

September 5, 2025

Keep in mind there are *two* PDFs available (of which this is the latter):

1. a PDF of examinable material used as lecture slides, and
2. a PDF of non-examinable, extra material:
   - the associated notes page may be pre-populated with extra, written explaination of material covered in lecture(s), plus
   - anything with a "grey'ed out" header/footer represents extra material which is useful and/or interesting but out of scope (and hence not covered).

Notes:

Notes:

- Agenda:
  1. bridge gap between register machines and real-world micro-processors, and
  2. introduce various design *paradigms*,

  via two case-studies.

Notes:

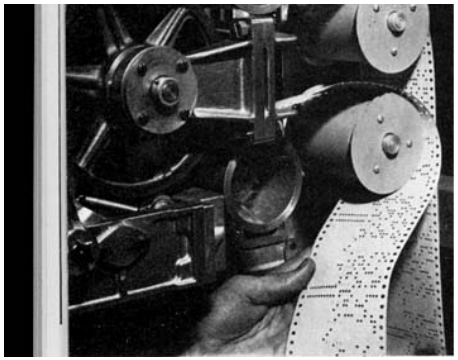


## Part 1: ASCC: a Harvard architecture (1)

- Example: **Automatic Sequence Controlled Calculator (ASCC)** [3].
  - Designed by Aiken and IBM; installed at Harvard University circa 1944,
  - 23.0m$^2$ footprint; 4.3t weight,
  - 72-element, 23-digit memory (i.e., decimal representation),
  - upto $\sim$ 3 operations per-second,
  - programs read from paper tape via a tape reader.

Notes:

- Example: **Automatic Sequence Controlled Calculator (ASCC)** [3].

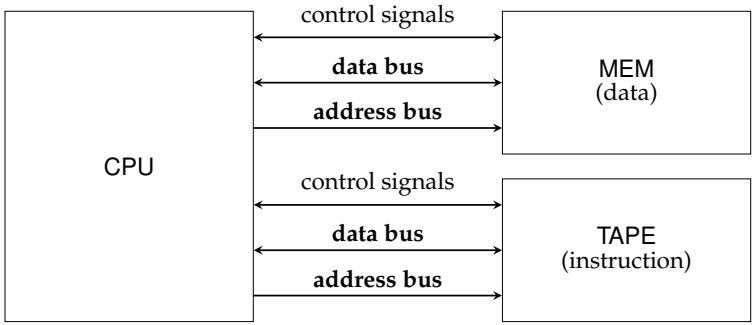


Notes:



Part 1: ASCC: a Harvard architecture (1)

- Example: **Automatic Sequence Controlled Calculator (ASCC)** [3].
  - consider an imaginary, ASCC-*like* design which
    - is similar to an accumulator machine, so has an accumulator called A,
    - uses a 6-digit decimal representation of values,
    - understands instructions from some set,
  - this design is an example of a **Harvard architecture**,
  - i.e., there are two, *separate* data and instruction memories:




  - the program, which is a sequence of instructions, is stored on TAPE.

Notes:

## Example (instruction set)

- nop, i.e., do nothing.
- halt, i.e., halt or stop execution.
- $\mathsf{A} \leftarrow n$, i.e., load the number $n$ into the accumulator A.
- $\mathsf{MEM}[n] \leftarrow \mathsf{A}$, i.e., store the number in accumulator A into address $n$ of the memory.
- $\mathsf{A} \leftarrow \mathsf{MEM}[n]$, i.e., load the number in address $n$ in memory into the accumulator A.
- $\mathsf{A} \leftarrow \mathsf{A} + \mathsf{MEM}[n]$, i.e., add the number in address $n$ of the memory to the accumulator A and store the result back in the accumulator.
- $\mathsf{A} \leftarrow \mathsf{A} - \mathsf{MEM}[n]$, i.e., subtract the number in address $n$ of the memory from the accumulator A and store the result back in the accumulator.
- $\mathsf{A} \leftarrow \mathsf{A} \oplus \mathsf{MEM}[n]$, i.e., XOR the number in address $n$ of the memory with the accumulator A and store the result back in the accumulator.

Notes:





## Algorithm (fetch-decode-execute cycle)

1. Encode a program $P$ onto paper tape; load this tape into the tape reader, then zero A and start the computer.
2. Using the tape reader, fetch the next instruction in the program and store it in IR.
3. If
   - 3.1 IR $= \perp$ (i.e., an invalid instruction is encountered), or
   - 3.2 IR = halt (i.e., the program halts normally)

   then halt the computer, otherwise execute IR (i.e., perform the operation it specifies).
4. Repeat from step 2.

Notes:

- Note that one cannot alter the program once execution begins, and there is no way to alter control-flow; instructions are executed in the same order they are written on the tape.

## Example (compute 10 + 20)

| CPU | | |
|---|---|---|
| state | = | reset |
| IR | = | |
| A | = | 0 |

| MEM | |
|---|---|
| Address | Value |
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 10 |
| 5 | 20 |
| 6 | 0 |
| 7 | 0 |

| TAPE | |
|---|---|
| Address | Semantics |
| 0 | A ← MEM[4] |
| 1 | A ← A + MEM[5] |
| 2 | MEM[6] ← A |
| 3 | halt |
| 4 | nop |
| 5 | nop |
| 6 | nop |
| 7 | nop |

Notes:



Part 1: ASCC: a Harvard architecture (4)

## Example (compute 10 + 20)

| CPU | | |
|---|---|---|
| state | = | fetch |
| IR | = | A ← MEM[4] |
| A | = | 0 |

| MEM | |
|---|---|
| Address | Value |
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 10 |
| 5 | 20 |
| 6 | 0 |
| 7 | 0 |

| TAPE | |
|---|---|
| Address | Semantics |
| 0 | A ← MEM[4] |
| 1 | A ← A + MEM[5] |
| 2 | MEM[6] ← A |
| 3 | halt |
| 4 | nop |
| 5 | nop |
| 6 | nop |
| 7 | nop |

Notes:

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (compute 10 + 20)

| CPU | | |
|---|---|---|
| state | = | execute |
| IR | = | A ← MEM[4] |
| A | = | 10 |

| MEM | |
|---|---|
| Address | Value |
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 10 |
| 5 | 20 |
| 6 | 0 |
| 7 | 0 |

| TAPE | |
|---|---|
| Address | Semantics |
| 0 | A ← MEM[4] |
| 1 | A ← A + MEM[5] |
| 2 | MEM[6] ← A |
| 3 | halt |
| 4 | nop |
| 5 | nop |
| 6 | nop |
| 7 | nop |

Notes:



Part 1: ASCC: a Harvard architecture (4)

## Example (compute 10 + 20)

| CPU | | |
|---|---|---|
| state | = | fetch |
| IR | = | A ← A + MEM[5] |
| A | = | 10 |

| MEM | |
|---|---|
| Address | Value |
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 10 |
| 5 | 20 |
| 6 | 0 |
| 7 | 0 |

| TAPE | |
|---|---|
| Address | Semantics |
| 0 | A ← MEM[4] |
| 1 | A ← A + MEM[5] |
| 2 | MEM[6] ← A |
| 3 | halt |
| 4 | nop |
| 5 | nop |
| 6 | nop |
| 7 | nop |

Notes:

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (compute 10 + 20)

| CPU | | |
|---|---|---|
| state | = | execute |
| IR | = | A ← A + MEM[5] |
| A | = | 30 |

| MEM | |
|---|---|
| Address | Value |
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 10 |
| 5 | 20 |
| 6 | 0 |
| 7 | 0 |

| TAPE | |
|---|---|
| Address | Semantics |
| 0 | A ← MEM[4] |
| 1 | A ← A + MEM[5] |
| 2 | MEM[6] ← A |
| 3 | halt |
| 4 | nop |
| 5 | nop |
| 6 | nop |
| 7 | nop |

Notes:



Part 1: ASCC: a Harvard architecture (4)

## Example (compute 10 + 20)

| CPU | | |
|---|---|---|
| state | = | fetch |
| IR | = | MEM[6] ← A |
| A | = | 30 |

| MEM | |
|---|---|
| Address | Value |
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 10 |
| 5 | 20 |
| 6 | 0 |
| 7 | 0 |

| TAPE | |
|---|---|
| Address | Semantics |
| 0 | A ← MEM[4] |
| 1 | A ← A + MEM[5] |
| 2 | MEM[6] ← A |
| 3 | halt |
| 4 | nop |
| 5 | nop |
| 6 | nop |
| 7 | nop |

Notes:

## Example (compute 10 + 20)

| CPU | | |
|---|---|---|
| state | = | execute |
| IR | = | MEM[6] ← A |
| A | = | 30 |

| MEM | |
|---|---|
| Address | Value |
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 10 |
| 5 | 20 |
| 6 | 30 |
| 7 | 0 |

| TAPE | |
|---|---|
| Address | Semantics |
| 0 | A ← MEM[4] |
| 1 | A ← A + MEM[5] |
| 2 | MEM[6] ← A |
| 3 | halt |
| 4 | nop |
| 5 | nop |
| 6 | nop |
| 7 | nop |

Notes:



Part 1: ASCC: a Harvard architecture (4)

Example (compute 10 + 20)

| CPU | | |
|---|---|---|
| state | = | fetch |
| IR | = | halt |
| A | = | 30 |

| MEM | |
|---|---|
| Address | Value |
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 10 |
| 5 | 20 |
| 6 | 30 |
| 7 | 0 |

| TAPE | |
|---|---|
| Address | Semantics |
| 0 | A ← MEM[4] |
| 1 | A ← A + MEM[5] |
| 2 | MEM[6] ← A |
| 3 | halt |
| 4 | nop |
| 5 | nop |
| 6 | nop |
| 7 | nop |

Notes:

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (compute 10 + 20)

| CPU | | |
|---|---|---|
| state | = | execute |
| IR | = | halt |
| A | = | 30 |

| MEM | |
|---|---|
| Address | Value |
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 10 |
| 5 | 20 |
| 6 | 30 |
| 7 | 0 |

| TAPE | |
|---|---|
| Address | Semantics |
| 0 | A ← MEM[4] |
| 1 | A ← A + MEM[5] |
| 2 | MEM[6] ← A |
| 3 | halt |
| 4 | nop |
| 5 | nop |
| 6 | nop |
| 7 | nop |

Notes:



## An Aside: how imaginary is imaginary?

- Example:
  - Released circa 1980, the **Intel 8051** [4] is a
    - 8-bit (i.e., has an 8-bit ALU and accumulator A),
    - Harvard architecture,

    micro-controller, with upto
    - 256B of internal data memory (or IRAM),
    - 64kB of external data memory (or XRAM), and
    - 64kB of program memory (or PRAM).
  - The 8051 instruction set isn't *too* far off what we have, e.g.,

    | | | |
    |---|---|---|
    | `NOP` | ↦ | |
    | `MOV x` | ↦ | A ← `x` |
    | `MOV y` | ↦ | A ← MEM[y] |
    | `MOV y` | ↦ | MEM[y] ← A |
    | `ADDC y` | ↦ | A ← A + MEM[y] + carry |
    | `SUBB y` | ↦ | A ← A − MEM[y] − carry |
    | `XRL y` | ↦ | A ← A ⊕ MEM[y] |

    where
    - `x` is an 8-bit value, and `y` is an address in IRAM, and
    - there are a *variety* of different versions of `MOV`.

Notes:

## Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (1)

Hang on, why have two *separate* memories [14]: we can have one *unified* memory, and use it to store both data *and* data that represents instructions (i.e., encoded instructions).

https://en.wikipedia.org/wiki/File:JohnvonNeumann-LosAlamos.gif



Notes:

- There's some debate about who had the idea first, and even precise definition what the stored-program concept constitutes. Haigh [12] presents a detailed treatment, but, for example
  1. Zuse described the same concept, circa 1936, in a patent application for his range of early computers,
  2. von Neumann worked on the EDVAC project with Mauchly and Eckert, circa 1945, who previously built the ENIAC, and
  3. Turing described a stored-program computer, the Pilot ACE, circa 1946.

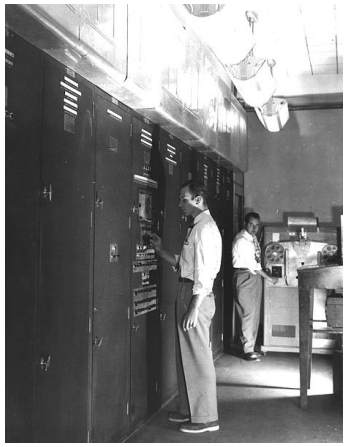
## Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (2)

- Example: **Electronic Discrete Variable Automatic Computer (EDVAC)** [1].
  - Designed by Mauchly and Eckert; installed at US-based BRL circa 1949,
  - 45.0m$^2$ footprint; 7.8t weight,
  - 1000-element, 44-bit memory (i.e., binary representation),
  - upto $\sim$ 1160 operations per-second,
  - programs read from magnetic tape into memory via a tape reader.



Notes:

- Example: **Electronic Discrete Variable Automatic Computer (EDVAC)** [1].

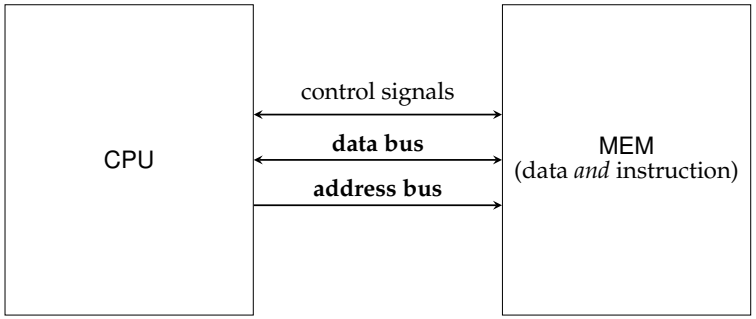


Notes:



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (2)

- Example: **Electronic Discrete Variable Automatic Computer (EDVAC)** [1].
  - consider an imaginary, EDVAC-*like* design which
    - inherits the previous ASCC-like design,
    - adds a program counter called PC,
    - updates the instruction set, e.g., to allow control of PC and define an instruction encoding,
    - updates the fetch-decode-execute cycle to match.
  - this design is an example of a **Princeton architecture** (aka. **von Neumann architecture**),
  - i.e., there is one, *unified* data and instruction memory:




  - the program, which is a sequence of instructions, is stored on MEM.

Notes:

## Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (3)

### Example (instruction set)

- 00*nnnn* means nop.
- 10*nnnn* means halt.
- 20*nnnn* means A $\leftarrow n$.
- 21*nnnn* means MEM[$n$] $\leftarrow$ A.
- 22*nnnn* means A $\leftarrow$ MEM[$n$].
- 30*nnnn* means A $\leftarrow$ A + MEM[$n$].
- 31*nnnn* means A $\leftarrow$ A – MEM[$n$].
- 32*nnnn* means A $\leftarrow$ A $\oplus$ MEM[$n$].
- 40*nnnn* means PC $\leftarrow n$.
- 41*nnnn* means PC $\leftarrow n$ iff. A = 0.
- 42*nnnn* means PC $\leftarrow n$ iff. A $\neq$ 0.

Notes:



## Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (4)

### Algorithm (fetch-decode-execute cycle)

1. Encode a program $P$ onto magnetic tape; load this tape into memory using the tape reader, then zero A and PC and start the computer.
2. From the address in PC, fetch the next instruction in the program and store it in IR.
3. Increment PC so it points to the next instruction.
4. If
   - 4.1 IR $=\perp$ (i.e., an invalid instruction is encountered), or
   - 4.2 IR = halt (i.e., the program halts normally)

   then halt the computer, otherwise execute IR (i.e., perform the operation it specifies).
5. Repeat from step 2.

Notes:

## Example (compute 10 + 20)

| CPU | | |
|---|---|---|
| state | = | reset |
| PC | = | 0 |
| IR | = | |
| | = | |
| A | = | 0 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 100000 | halt |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

Notes:



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (5)

## Example (compute 10 + 20)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 0 |
| IR | = | 220004 |
| | = | |
| A | = | 0 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 100000 | halt |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

Notes:

## Example (compute 10 + 20)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 1 |
| IR | = | 220004 |
| | = | A ← MEM[4] |
| A | = | 0 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 100000 | halt |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

Notes:



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (5)

## Example (compute 10 + 20)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 1 |
| IR | = | 220004 |
| | = | A ← MEM[4] |
| A | = | 10 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 100000 | halt |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

Notes:

## Example (compute 10 + 20)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 1 |
| IR | = | 300005 |
| | = | |
| A | = | 10 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 100000 | halt |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

Notes:



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (5)

## Example (compute 10 + 20)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 2 |
| IR | = | 300005 |
| | = | A ← A + MEM[5] |
| A | = | 10 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 100000 | halt |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

Notes:

## Example (compute 10 + 20)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 2 |
| IR | = | 300005 |
| | = | A ← A + MEM[5] |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 100000 | halt |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

Notes:



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (5)

## Example (compute 10 + 20)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 2 |
| IR | = | 210006 |
| | = | |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 100000 | halt |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

Notes:

## Example (compute 10 + 20)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 3 |
| IR | = | 210006 |
| | = | MEM[6] ← A |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 100000 | halt |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

Notes:



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (5)

## Example (compute 10 + 20)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 3 |
| IR | = | 210006 |
| | = | MEM[6] ← A |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 100000 | halt |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 30 | nop |
| 7 | 0 | nop |

Notes:

## Example (compute 10 + 20)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 3 |
| IR | = | 100000 |
| | = | |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 100000 | halt |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 30 | nop |
| 7 | 0 | nop |

Notes:



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (5)

## Example (compute 10 + 20)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 4 |
| IR | = | 100000 |
| | = | halt |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 100000 | halt |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 30 | nop |
| 7 | 0 | nop |

Notes:

## Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (5)

### Example (compute 10 + 20)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 4 |
| IR | = | 100000 |
| | = | halt |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 100000 | halt |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 30 | nop |
| 7 | 0 | nop |

Notes:



## Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (6)

### Example (compute 10 + 20, including an infinite loop)

| CPU | | |
|---|---|---|
| state | = | reset |
| PC | = | 0 |
| IR | = | |
| | = | |
| A | = | 0 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 400000 | PC ← 0 |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

$$\text{stored program} \Rightarrow \text{implication \#1} = \begin{cases} 1. \text{ we can control PC, } \textit{so} \\ 2. \text{ we can, e.g., write loops.} \end{cases}$$

Notes:

## Example (compute 10 + 20, including an infinite loop)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 0 |
| IR | = | 220004 |
| | = | |
| A | = | 0 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 400000 | PC ← 0 |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #1 = { 1. we can control PC, *so* 2. we can, e.g., write loops.

Notes:



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (6)

## Example (compute 10 + 20, including an infinite loop)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 1 |
| IR | = | 220004 |
| | = | A ← MEM[4] |
| A | = | 0 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 400000 | PC ← 0 |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #1 = { 1. we can control PC, *so* 2. we can, e.g., write loops.

Notes:

## Example (compute 10 + 20, including an infinite loop)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 1 |
| IR | = | 220004 |
| | = | A ← MEM[4] |
| A | = | 10 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 400000 | PC ← 0 |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #1 = { 1. we can control PC, *so* 2. we can, e.g., write loops.

Notes:



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (6)

## Example (compute 10 + 20, including an infinite loop)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 1 |
| IR | = | 300005 |
| | = | |
| A | = | 10 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 400000 | PC ← 0 |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #1 = { 1. we can control PC, *so* 2. we can, e.g., write loops.

Notes:

© Daniel Page ⟨cdp@cs.bris.ac.uk⟩ Computer Architecture University of BRISTOL git # b282dbb9 @ 2025-09-03

## Example (compute 10 + 20, including an infinite loop)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 2 |
| IR | = | 300005 |
| | = | A ← A + MEM[5] |
| A | = | 10 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 400000 | PC ← 0 |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #1 = { 1. we can control PC, *so* 2. we can, e.g., write loops.

Notes:



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (6)

## Example (compute 10 + 20, including an infinite loop)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 2 |
| IR | = | 300005 |
| | = | A ← A + MEM[5] |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 400000 | PC ← 0 |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #1 = { 1. we can control PC, *so* 2. we can, e.g., write loops.

Notes:

© Daniel Page ⟨cdp@cs.bris.ac.uk⟩ Computer Architecture University of BRISTOL git # b282dbb9 @ 2025-09-03

## Example (compute 10 + 20, including an infinite loop)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 2 |
| IR | = | 210006 |
| | = | |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 400000 | PC ← 0 |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #1 = { 1. we can control PC, *so* 2. we can, e.g., write loops.

Notes:



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (6)

## Example (compute 10 + 20, including an infinite loop)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 3 |
| IR | = | 210006 |
| | = | MEM[6] ← A |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 400000 | PC ← 0 |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #1 = { 1. we can control PC, *so* 2. we can, e.g., write loops.

Notes:

© Daniel Page ⟨cdp@cs.bris.ac.uk⟩ Computer Architecture University of BRISTOL git # b282dbb9 @ 2025-09-03

## Example (compute 10 + 20, including an infinite loop)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 3 |
| IR | = | 210006 |
| | = | MEM[6] ← A |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 400000 | PC ← 0 |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 30 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #1 = { 1. we can control PC, *so* 2. we can, e.g., write loops.

Notes:



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (6)

## Example (compute 10 + 20, including an infinite loop)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 3 |
| IR | = | 400000 |
| | = | |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 400000 | PC ← 0 |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 30 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #1 = { 1. we can control PC, *so* 2. we can, e.g., write loops.

Notes:

© Daniel Page ⟨cdp@cs.bris.ac.uk⟩ Computer Architecture University of BRISTOL git # b282dbb9 @ 2025-09-03

## Example (compute 10 + 20, including an infinite loop)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 4 |
| IR | = | 400000 |
| | = | PC ← 0 |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 400000 | PC ← 0 |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 30 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #1 = { 1. we can control PC, *so* 2. we can, e.g., write loops.

Notes:



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (6)

## Example (compute 10 + 20, including an infinite loop)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 0 |
| IR | = | 400000 |
| | = | PC ← 0 |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 400000 | PC ← 0 |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 30 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #1 = { 1. we can control PC, *so* 2. we can, e.g., write loops.

Notes:

© Daniel Page ⟨cdp@cs.bris.ac.uk⟩ Computer Architecture University of BRISTOL git # b282dbb9 @ 2025-09-03

## Example (compute 10 + 20, including an infinite loop)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 0 |
| IR | = | 220004 |
| | = | |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 400000 | PC ← 0 |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 30 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #1 = { 1. we can control PC, *so* 2. we can, e.g., write loops.

Notes:



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (6)

## Example (compute 10 + 20, including an infinite loop)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 1 |
| IR | = | 220004 |
| | = | A ← MEM[4] |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 400000 | PC ← 0 |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 30 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #1 = { 1. we can control PC, *so* 2. we can, e.g., write loops.

Notes:

© Daniel Page ⟨cdp@cs.bris.ac.uk⟩ Computer Architecture University of BRISTOL git # b282dbb9 @ 2025-09-03

## Example (compute 10 + 20, including an infinite loop)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 1 |
| IR | = | 220004 |
| | = | A ← MEM[4] |
| A | = | 10 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210006 | MEM[6] ← A |
| 3 | 400000 | PC ← 0 |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 30 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #1 = { 1. we can control PC, *so* 2. we can, e.g., write loops.

Notes:





## Example (compute 10 + 20, including programming mistake)

| CPU | | |
|---|---|---|
| state | = | reset |
| PC | = | 0 |
| IR | = | |
| | = | |
| A | = | 0 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210003 | MEM[3] ← A |
| 3 | 100000 | halt |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #2 = { 1. data *and* instructions are stored in MEM, *so* 2. we can, e.g., write *self-modifying* code.

Notes:

## Example (compute 10 + 20, including programming mistake)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 0 |
| IR | = | 220004 |
| | = | |
| A | = | 0 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210003 | MEM[3] ← A |
| 3 | 100000 | halt |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #2 = { 1. data *and* instructions are stored in MEM, *so* 2. we can, e.g., write *self-modifying* code.

Notes:



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (7)

Example (compute 10 + 20, including programming mistake)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 1 |
| IR | = | 220004 |
| | = | A ← MEM[4] |
| A | = | 0 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210003 | MEM[3] ← A |
| 3 | 100000 | halt |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #2 = { 1. data *and* instructions are stored in MEM, *so* 2. we can, e.g., write *self-modifying* code.

Notes:

© Daniel Page ⟨cs@phoo.cf⟩ Computer Architecture — University of BRISTOL — git # b282dbb9 @ 2025-09-03

## Example (compute 10 + 20, including programming mistake)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 1 |
| IR | = | 220004 |
| | = | A ← MEM[4] |
| A | = | 10 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210003 | MEM[3] ← A |
| 3 | 100000 | halt |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #2 = { 1. data *and* instructions are stored in MEM, *so* 2. we can, e.g., write *self-modifying* code.

Notes:



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (7)

## Example (compute 10 + 20, including programming mistake)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 1 |
| IR | = | 300005 |
| | = | |
| A | = | 10 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210003 | MEM[3] ← A |
| 3 | 100000 | halt |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #2 = { 1. data *and* instructions are stored in MEM, *so* 2. we can, e.g., write *self-modifying* code.

Notes:

© Daniel Page ⟨cs@cs.bris.ac.uk⟩ Computer Architecture — University of BRISTOL — git # b282dbb9 @ 2025-09-03

## Example (compute 10 + 20, including programming mistake)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 2 |
| IR | = | 300005 |
| | = | A ← A + MEM[5] |
| A | = | 10 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210003 | MEM[3] ← A |
| 3 | 100000 | halt |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #2 = { 1. data *and* instructions are stored in MEM, *so* 2. we can, e.g., write *self-modifying* code.

Notes:



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (7)

## Example (compute 10 + 20, including programming mistake)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 2 |
| IR | = | 300005 |
| | = | A ← A + MEM[5] |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210003 | MEM[3] ← A |
| 3 | 100000 | halt |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #2 = { 1. data *and* instructions are stored in MEM, *so* 2. we can, e.g., write *self-modifying* code.

Notes:

## Example (compute 10 + 20, including programming mistake)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 2 |
| IR | = | 210003 |
| | = | |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210003 | MEM[3] ← A |
| 3 | 100000 | halt |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #2 = { 1. data *and* instructions are stored in MEM, *so* 2. we can, e.g., write *self-modifying* code.

Notes:



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (7)

## Example (compute 10 + 20, including programming mistake)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 3 |
| IR | = | 210003 |
| | = | MEM[3] ← A |
| A | = | 30 |


| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210003 | MEM[3] ← A |
| 3 | 100000 | halt |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #2 = { 1. data *and* instructions are stored in MEM, *so* 2. we can, e.g., write *self-modifying* code.


Notes:

## Example (compute 10 + 20, including programming mistake)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 3 |
| IR | = | 210003 |
| | = | MEM[3] ← A |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210003 | MEM[3] ← A |
| 3 | 30 | nop |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #2 = { 1. data *and* instructions are stored in MEM, *so* 2. we can, e.g., write *self-modifying* code.

Notes:



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (7)

## Example (compute 10 + 20, including programming mistake)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 3 |
| IR | = | 000030 |
| | = | |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210003 | MEM[3] ← A |
| 3 | 30 | nop |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #2 = { 1. data *and* instructions are stored in MEM, *so* 2. we can, e.g., write *self-modifying* code.

Notes:

## Example (compute 10 + 20, including programming mistake)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 4 |
| IR | = | 000030 |
| | = | nop |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210003 | MEM[3] ← A |
| 3 | 30 | nop |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #2 = { 1. data *and* instructions are stored in MEM, *so* 2. we can, e.g., write *self-modifying* code.

Notes:



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (7)

Example (compute 10 + 20, including programming mistake)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 4 |
| IR | = | 000030 |
| | = | nop |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210003 | MEM[3] ← A |
| 3 | 30 | nop |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #2 = { 1. data *and* instructions are stored in MEM, *so* 2. we can, e.g., write *self-modifying* code.

Notes:

© Daniel Page ⟨cs@phoo.cs.bris.ac.uk⟩ Computer Architecture University of BRISTOL git # b282dbb9 @ 2025-09-03

## Example (compute 10 + 20, including programming mistake)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 4 |
| IR | = | 000010 |
| | = | |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210003 | MEM[3] ← A |
| 3 | 30 | nop |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #2 = { 1. data *and* instructions are stored in MEM, *so* 2. we can, e.g., write *self-modifying* code.

Notes:



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (7)

## Example (compute 10 + 20, including programming mistake)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 5 |
| IR | = | 000010 |
| | = | nop |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210003 | MEM[3] ← A |
| 3 | 30 | nop |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #2 = { 1. data *and* instructions are stored in MEM, *so* 2. we can, e.g., write *self-modifying* code.

Notes:

## Example (compute 10 + 20, including programming mistake)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 5 |
| IR | = | 000010 |
| | = | nop |
| A | = | 30 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220004 | A ← MEM[4] |
| 1 | 300005 | A ← A + MEM[5] |
| 2 | 210003 | MEM[3] ← A |
| 3 | 30 | nop |
| 4 | 10 | nop |
| 5 | 20 | nop |
| 6 | 0 | nop |
| 7 | 0 | nop |

stored program ⇒ implication #2 = { 1. data *and* instructions are stored in MEM, *so* 2. we can, e.g., write *self-modifying* code.

Notes:





- Beware: self-modifying code isn't *just* a destructive "problem".
- Problem: write a program that increments each element in a 3-element sequence called $X$.

Notes:

- Beware: self-modifying code isn't *just* a destructive "problem".

- Solution: use self-modifying code constructively.
  - In C, to load the $i$-th element `X[ i ]` we and load from address `&X+i`.
  - We don't have an instruction that can load from MEM[`&X+i`], only one that loads from MEM[$n$] where $n$ is *fixed* at compile-time!
  - So *a* solution is to *modify* an instruction that accessed MEM[$n$] in the $i$-th iteration, so in the $(i+1)$-th iteration it accesses MEM[$n+1$].

Notes:





**Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)**

| CPU | | |
|---|---|---|
| state | = | reset |
| PC | = | 0 |
| IR | = | |
| | = | |
| A | = | 0 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220015 | A ← MEM[15] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 0 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 0 |
| IR | = | 220015 |
| | = | |
| A | = | 0 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220015 | A ← MEM[15] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 0 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 1 |
| IR | = | 220015 |
| | = | A ← MEM[15] |
| A | = | 0 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220015 | A ← MEM[15] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 0 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 1 |
| IR | = | 220015 |
| | = | A ← MEM[15] |
| A | = | 0 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220015 | A ← MEM[15] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 0 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 1 |
| IR | = | 300013 |
| | = | |
| A | = | 0 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220015 | A ← MEM[15] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 0 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 2 |
| IR | = | 300013 |
| | = | A ← A + MEM[13] |
| A | = | 0 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220015 | A ← MEM[15] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 0 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 2 |
| IR | = | 300013 |
| | = | A ← A + MEM[13] |
| A | = | 1 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220015 | A ← MEM[15] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 0 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 2 |
| IR | = | 210015 |
| | = | |
| A | = | 1 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220015 | A ← MEM[15] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 0 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 3 |
| IR | = | 210015 |
| | = | MEM[15] ← A |
| A | = | 1 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220015 | A ← MEM[15] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 0 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 3 |
| IR | = | 210015 |
| | = | MEM[15] ← A |
| A | = | 1 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220015 | A ← MEM[15] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 3 |
| IR | = | 220000 |
| | = | |
| A | = | 1 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220015 | A ← MEM[15] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 4 |
| IR | = | 220000 |
| | = | A ← MEM[0] |
| A | = | 1 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220015 | A ← MEM[15] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 4 |
| IR | = | 220000 |
| | = | A ← MEM[0] |
| A | = | 220015 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220015 | A ← MEM[15] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 4 |
| IR | = | 300013 |
| | = | |
| A | = | 220015 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220015 | A ← MEM[15] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 5 |
| IR | = | 300013 |
| | = | A ← A + MEM[13] |
| A | = | 220015 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220015 | A ← MEM[15] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 5 |
| IR | = | 300013 |
| | = | A ← A + MEM[13] |
| A | = | 220016 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220015 | A ← MEM[15] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 5 |
| IR | = | 210000 |
| | = | |
| A | = | 220016 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220015 | A ← MEM[15] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 6 |
| IR | = | 210000 |
| | = | MEM[0] ← A |
| A | = | 220016 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220015 | A ← MEM[15] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 6 |
| IR | = | 210000 |
| | = | MEM[0] ← A |
| A | = | 220016 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 6 |
| IR | = | 220002 |
| | = | |
| A | = | 220016 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 7 |
| IR | = | 220002 |
| | = | A ← MEM[2] |
| A | = | 220016 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 7 |
| IR | = | 220002 |
| | = | A ← MEM[2] |
| A | = | 210015 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 7 |
| IR | = | 300013 |
| | = | |
| A | = | 210015 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 8 |
| IR | = | 300013 |
| | = | A ← A + MEM[13] |
| A | = | 210015 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 8 |
| IR | = | 300013 |
| | = | A ← A + MEM[13] |
| A | = | 210016 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 8 |
| IR | = | 210002 |
| | = | |
| A | = | 210016 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 9 |
| IR | = | 210002 |
| | = | MEM[2] ← A |
| A | = | 210016 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210015 | MEM[15] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 9 |
| IR | = | 210002 |
| | = | MEM[2] ← A |
| A | = | 210016 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 9 |
| IR | = | 220000 |
| | = | |
| A | = | 210016 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 10 |
| IR | = | 220000 |
| | = | A ← MEM[0] |
| A | = | 210016 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 10 |
| IR | = | 220000 |
| | = | A ← MEM[0] |
| A | = | 220016 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 10 |
| IR | = | 310014 |
| | = | |
| A | = | 220016 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 11 |
| IR | = | 310014 |
| | = | A ← A − MEM[14] |
| A | = | 220016 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 11 |
| IR | = | 310014 |
| | = | A ← A – MEM[14] |
| A | = | –2 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A – MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 11 |
| IR | = | 420000 |
| | = | |
| A | = | –2 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A – MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 12 |
| IR | = | 420000 |
| | = | PC ← 0 iff. A ≠ 0 |
| A | = | −2 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 0 |
| IR | = | 420000 |
| | = | PC ← 0 iff. A ≠ 0 |
| A | = | −2 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 0 |
| IR | = | 220016 |
| | = | |
| A | = | −2 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 1 |
| IR | = | 220016 |
| | = | A ← MEM[16] |
| A | = | −2 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 1 |
| IR | = | 220016 |
| | = | A ← MEM[16] |
| A | = | 1 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 1 |
| IR | = | 300013 |
| | = | |
| A | = | 1 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 2 |
| IR | = | 300013 |
| | = | A ← A + MEM[13] |
| A | = | 1 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 2 |
| IR | = | 300013 |
| | = | A ← A + MEM[13] |
| A | = | 2 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 2 |
| IR | = | 210016 |
| | = | |
| A | = | 2 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 3 |
| IR | = | 210016 |
| | = | MEM[16] ← A |
| A | = | 2 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 1 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 3 |
| IR | = | 210016 |
| | = | MEM[16] ← A |
| A | = | 2 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 3 |
| IR | = | 220000 |
| | = | |
| A | = | 2 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 4 |
| IR | = | 220000 |
| | = | A ← MEM[0] |
| A | = | 2 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 4 |
| IR | = | 220000 |
| | = | A ← MEM[0] |
| A | = | 220016 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 4 |
| IR | = | 300013 |
| | = | |
| A | = | 220016 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 5 |
| IR | = | 300013 |
| | = | A ← A + MEM[13] |
| A | = | 220016 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 5 |
| IR | = | 300013 |
| | = | A ← A + MEM[13] |
| A | = | 220017 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 5 |
| IR | = | 210000 |
| | = | |
| A | = | 220017 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 6 |
| IR | = | 210000 |
| | = | MEM[0] ← A |
| A | = | 220017 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220016 | A ← MEM[16] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 6 |
| IR | = | 210000 |
| | = | MEM[0] ← A |
| A | = | 220017 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 6 |
| IR | = | 220002 |
| | = | |
| A | = | 220017 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 7 |
| IR | = | 220002 |
| | = | A ← MEM[2] |
| A | = | 220017 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 7 |
| IR | = | 220002 |
| | = | A ← MEM[2] |
| A | = | 210016 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 7 |
| IR | = | 300013 |
| | = | |
| A | = | 210016 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 8 |
| IR | = | 300013 |
| | = | A ← A + MEM[13] |
| A | = | 210016 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 8 |
| IR | = | 300013 |
| | = | A ← A + MEM[13] |
| A | = | 210017 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 8 |
| IR | = | 210002 |
| | = | |
| A | = | 210017 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 9 |
| IR | = | 210002 |
| | = | MEM[2] ← A |
| A | = | 210017 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210016 | MEM[16] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 9 |
| IR | = | 210002 |
| | = | MEM[2] ← A |
| A | = | 210017 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 9 |
| IR | = | 220000 |
| | = | |
| A | = | 210017 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 10 |
| IR | = | 220000 |
| | = | A ← MEM[0] |
| A | = | 210017 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 10 |
| IR | = | 220000 |
| | = | A ← MEM[0] |
| A | = | 220017 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 10 |
| IR | = | 310014 |
| | = | |
| A | = | 220017 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 11 |
| IR | = | 310014 |
| | = | A ← A − MEM[14] |
| A | = | 220017 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 11 |
| IR | = | 310014 |
| | = | A ← A – MEM[14] |
| A | = | −1 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A – MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 11 |
| IR | = | 420000 |
| | = | |
| A | = | −1 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A – MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 12 |
| IR | = | 420000 |
| | = | PC ← 0 iff. A ≠ 0 |
| A | = | −1 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 0 |
| IR | = | 420000 |
| | = | PC ← 0 iff. A ≠ 0 |
| A | = | −1 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 0 |
| IR | = | 220017 |
| | = | |
| A | = | −1 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 1 |
| IR | = | 220017 |
| | = | A ← MEM[17] |
| A | = | −1 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 1 |
| IR | = | 220017 |
| | = | A ← MEM[17] |
| A | = | 2 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 1 |
| IR | = | 300013 |
| | = | |
| A | = | 2 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 2 |
| IR | = | 300013 |
| | = | A ← A + MEM[13] |
| A | = | 2 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 2 |
| IR | = | 300013 |
| | = | A ← A + MEM[13] |
| A | = | 3 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 2 |
| IR | = | 210017 |
| | = | |
| A | = | 3 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 3 |
| IR | = | 210017 |
| | = | MEM[17] ← A |
| A | = | 3 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 2 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 3 |
| IR | = | 210017 |
| | = | MEM[17] ← A |
| A | = | 3 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 3 |
| IR | = | 220000 |
| | = | |
| A | = | 3 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 4 |
| IR | = | 220000 |
| | = | A ← MEM[0] |
| A | = | 3 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 4 |
| IR | = | 220000 |
| | = | A ← MEM[0] |
| A | = | 220017 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 4 |
| IR | = | 300013 |
| | = | |
| A | = | 220017 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 5 |
| IR | = | 300013 |
| | = | A ← A + MEM[13] |
| A | = | 220017 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 5 |
| IR | = | 300013 |
| | = | A ← A + MEM[13] |
| A | = | 220018 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 5 |
| IR | = | 210000 |
| | = | |
| A | = | 220018 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 6 |
| IR | = | 210000 |
| | = | MEM[0] ← A |
| A | = | 220018 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220017 | A ← MEM[17] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 6 |
| IR | = | 210000 |
| | = | MEM[0] ← A |
| A | = | 220018 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220018 | A ← MEM[18] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 6 |
| IR | = | 220002 |
| | = | |
| A | = | 220018 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220018 | A ← MEM[18] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 7 |
| IR | = | 220002 |
| | = | A ← MEM[2] |
| A | = | 220018 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220018 | A ← MEM[18] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 7 |
| IR | = | 220002 |
| | = | A ← MEM[2] |
| A | = | 210017 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220018 | A ← MEM[18] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 7 |
| IR | = | 300013 |
| | = | |
| A | = | 210017 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220018 | A ← MEM[18] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 8 |
| IR | = | 300013 |
| | = | A ← A + MEM[13] |
| A | = | 210017 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220018 | A ← MEM[18] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 8 |
| IR | = | 300013 |
| | = | A ← A + MEM[13] |
| A | = | 210018 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220018 | A ← MEM[18] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 8 |
| IR | = | 210002 |
| | = | |
| A | = | 210018 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220018 | A ← MEM[18] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 9 |
| IR | = | 210002 |
| | = | MEM[2] ← A |
| A | = | 210018 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220018 | A ← MEM[18] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210017 | MEM[17] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 9 |
| IR | = | 210002 |
| | = | MEM[2] ← A |
| A | = | 210018 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220018 | A ← MEM[18] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210018 | MEM[18] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 9 |
| IR | = | 220000 |
| | = | |
| A | = | 210018 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220018 | A ← MEM[18] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210018 | MEM[18] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 10 |
| IR | = | 220000 |
| | = | A ← MEM[0] |
| A | = | 210018 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220018 | A ← MEM[18] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210018 | MEM[18] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 10 |
| IR | = | 220000 |
| | = | A ← MEM[0] |
| A | = | 220018 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220018 | A ← MEM[18] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210018 | MEM[18] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 10 |
| IR | = | 310014 |
| | = | |
| A | = | 220018 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220018 | A ← MEM[18] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210018 | MEM[18] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 11 |
| IR | = | 310014 |
| | = | A ← A − MEM[14] |
| A | = | 220018 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220018 | A ← MEM[18] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210018 | MEM[18] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 11 |
| IR | = | 310014 |
| | = | A ← A – MEM[14] |
| A | = | 0 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220018 | A ← MEM[18] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210018 | MEM[18] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A – MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 11 |
| IR | = | 420000 |
| | = | |
| A | = | 0 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220018 | A ← MEM[18] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210018 | MEM[18] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A – MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 12 |
| IR | = | 420000 |
| | = | PC ← 0 iff. A ≠ 0 |
| A | = | 0 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220018 | A ← MEM[18] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210018 | MEM[18] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 12 |
| IR | = | 420000 |
| | = | PC ← 0 iff. A ≠ 0 |
| A | = | 0 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220018 | A ← MEM[18] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210018 | MEM[18] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | fetch |
| PC | = | 12 |
| IR | = | 100000 |
| | = | |
| A | = | 0 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220018 | A ← MEM[18] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210018 | MEM[18] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (9)

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | decode |
| PC | = | 13 |
| IR | = | 100000 |
| | = | halt |
| A | = | 0 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220018 | A ← MEM[18] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210018 | MEM[18] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)

| CPU | | |
|---|---|---|
| state | = | execute |
| PC | = | 13 |
| IR | = | 100000 |
| | = | halt |
| A | = | 0 |

| MEM | | |
|---|---|---|
| Address | Value | Semantics |
| 0 | 220018 | A ← MEM[18] |
| 1 | 300013 | A ← A + MEM[13] |
| 2 | 210018 | MEM[18] ← A |
| 3 | 220000 | A ← MEM[0] |
| 4 | 300013 | A ← A + MEM[13] |
| 5 | 210000 | MEM[0] ← A |
| 6 | 220002 | A ← MEM[2] |
| 7 | 300013 | A ← A + MEM[13] |
| 8 | 210002 | MEM[2] ← A |
| 9 | 220000 | A ← MEM[0] |
| 10 | 310014 | A ← A − MEM[14] |
| 11 | 420000 | PC ← 0 iff. A ≠ 0 |
| 12 | 100000 | halt |
| 13 | 1 | nop |
| 14 | 220018 | A ← MEM[18] |
| 15 | 1 | nop |
| 16 | 2 | nop |
| 17 | 3 | nop |

Notes:

- The program is basically three parts
  1. MEM[0 . . . 2] increment the elements,
  2. MEM[3 . . . 8] self-modify the instructions in MEM[0] and MEM[2], and
  3. MEM[9 . . . 11] control iteration of the loop body,

  where MEM[13 . . . 14] are constants, and $X$ is held in MEM[15 . . . 17].



# Conclusions

## Comparison

A **Harvard architecture**

- has high(er) area wrt. use of 2 buses,
- has high(er) bandwidth wrt. use of 2 buses,
- implies less flexible, static memory utilisation,
- can specialise instruction vs. data accesses,
- disallows self-modifying code.

## Comparison

A **Princeton architecture** (aka. **von Neumann architecture**)

- has low(er) area wrt. use of 1 bus,
- has low(er) bandwidth wrt. use of 1 bus,
- implies more flexible, dynamic memory utilisation,
- cannot specialise instruction vs. data accesses,
- allows self-modifying code.

Notes:

## Definition

The term **memory wall** [13] refers to a gap between efficiency of instruction execution and memory access. If memory access latency and bandwidth are insufficient, the associated micro-processor may wait (i.e., becomes idle) until instructions and/or data are delivered: the efficiency of memory access will then limit the efficiency of instruction execution, vs. say clock frequency.

## Definition

The term **von Neumann bottleneck**, as introduced by Backus [11], is a criticism of

1. the stored-program concept, essentially using a similar argument as the term memory wall [13],
2. the intellectual bottleneck (or limitation) implied by programmers focusing on and optimising for von Neumann architectures.



Notes:

Conclusions

- Take away points:
  1. Both architectures are of historical, conceptual, *and* practical importance.
  2. Both architectures are viable options, but *both*
     - conceptual constraints, e.g., the memory wall, *and*
     - practical constraints, e.g., area, clock frequency, power consumption,

     highlight the need for intelligent design and implementation.
  3. Variants, e.g., so-called *modified* Harvard architectures, can act as an effective compromise.

© Daniel Page ⟨csdsp@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

Notes:

- A common modified Harvard architecture is where 1) there are separate paths for instruction and data access (per a Harvard architecture), but 2) said paths are backed by the same physical memory (per a Princeton architecture). For example, it is common to consider a memory hierarchy which includes separate instruction and data caches attached to the same, unified main memory.

# Additional Reading

- *Wikipedia: Harvard architecture*. URL: https://en.wikipedia.org/wiki/Harvard_architecture.
- *Wikipedia: von Neumann architecture*. URL: https://en.wikipedia.org/wiki/Von_Neumann_architecture.
- *Wikipedia: Self-modifying code*. URL: https://en.wikipedia.org/wiki/Self-modifying_code.
- *Wikipedia: Modified Harvard architecture*. URL: https://en.wikipedia.org/wiki/Modified_Harvard_architecture.
- D. Page. "Chapter 4: A functional and historical perspective". In: *A Practical Introduction to Computer Architecture*. 1st ed. Springer, 2009.
- A.S. Tanenbaum and T. Austin. "Section 1.2: Milestones in computer architecture". In: *Structured Computer Organisation*. 6th ed. Prentice Hall, 2012.
- W. Stallings. "Chapter 2: Computer evolution and performance". In: *Computer Organisation and Architecture*. 9th ed. Prentice Hall, 2013.

Notes:



# References

[1] *Wikipedia: Electronic Discrete Variable Automatic Computer (EDVAC)*. URL: https://en.wikipedia.org/wiki/EDVAC (see pp. 39, 41, 43).

[2] *Wikipedia: Harvard architecture*. URL: https://en.wikipedia.org/wiki/Harvard_architecture (see p. 373).

[3] *Wikipedia: Harvard Mark I*. URL: https://en.wikipedia.org/wiki/Harvard_Mark_I (see pp. 7, 9, 11).

[4] *Wikipedia: Intel MCS-51*. URL: https://en.wikipedia.org/wiki/Intel_MCS-51 (see p. 35).

[5] *Wikipedia: Modified Harvard architecture*. URL: https://en.wikipedia.org/wiki/Modified_Harvard_architecture (see p. 373).

[6] *Wikipedia: Self-modifying code*. URL: https://en.wikipedia.org/wiki/Self-modifying_code (see p. 373).

[7] *Wikipedia: von Neumann architecture*. URL: https://en.wikipedia.org/wiki/Von_Neumann_architecture (see p. 373).

[8] D. Page. "Chapter 4: A functional and historical perspective". In: *A Practical Introduction to Computer Architecture*. 1st ed. Springer, 2009 (see p. 373).

[9] W. Stallings. "Chapter 2: Computer evolution and performance". In: *Computer Organisation and Architecture*. 9th ed. Prentice Hall, 2013 (see p. 373).

[10] A.S. Tanenbaum and T. Austin. "Section 1.2: Milestones in computer architecture". In: *Structured Computer Organisation*. 6th ed. Prentice Hall, 2012 (see p. 373).

[11] J. Backus. "Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs". In: *Communications of the ACM (CACM)* 21.8 (1978), pp. 613–641 (see p. 369).

[12] T. Haigh. ""Stored Program Concept" Considered Harmful: History and Historiography". In: *Conference on Computability in Europe (CiE)*. LNCS 7921. Springer-Verlag, 2013, pp. 241–251 (see p. 38).

[13] W.A. Wulf and S.A. McKee. "Hitting the memory wall: implications of the obvious". In: *ACM SIGARCH Computer Architecture News* 23.1 (1995), pp. 20–24 (see p. 369).

[14] J. von Neumann. *First Draft of a Report on the EDVAC*. Tech. rep. 1945 (see p. 37).

Notes: