Computer Architecture

Daniel Page

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

October 14, 2024

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).

ı	Notes:
	Notes:
	Notes:
	Notes:
	Notes:

Notes:

COMS10015 lecture: week #11

► Agenda:

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

via two case-studies.



Part 1: ASCC: a Harvard architecture (1)

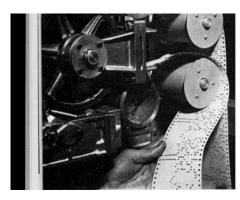
- **Example:** Automatic Sequence Controlled Calculator (ASCC) [3].
 - Designed by Aiken and IBM; installed at Harvard University circa 1944,

 - 23.0m² footprint; 4.3t weight,
 72-element, 23-digit memory (i.e., decimal representation),
 upto ~ 3 operations per-second,
 programs read from paper tape via a tape reader.

Notes:		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		ı
		l
Notes:		1
Notes:		

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



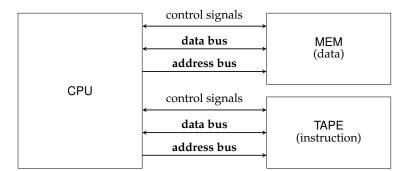


University of BRISTOL

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.
- ▶ A \leftarrow *n*, i.e., load the number *n* into the accumulator A.
- ▶ $MEM[n] \leftarrow A$, i.e., store the number in accumulator A into address n of the memory.
- ▶ A \leftarrow MEM[n], i.e., load the number in address n in memory into the accumulator A.
- ► A ← A + 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.
- A ← A − 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.
- ► A ← A ⊕ 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.

© Daniel Page (csdsp@bristol.ac Computer Architecture



git # c144985a @ 2024-10-14

Part 1: ASCC: a Harvard architecture (3)

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 = \bot (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:		
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)					
	N	ИEМ] [TAPE
	Address	Value	1	Address	Semantics
	0	0	1	0	$A \leftarrow MEM[4]$
CPU	1	0		1	$A \leftarrow A + MEM[5]$
state = reset	2	0		2	$MEM[6] \leftarrow A$
IR =	3	0		3	halt
A = 0	4	10		4	nop
	5	20		5	nop
	6	0		6	nop
	7	0		7	nop

© Daniel Page (categoristo Lacture) University of Computer Architecture University of BRISTOL git#c144985a@2024

Part 1: ASCC: a Harvard architecture (4)

Example (compute 10 + 20)				
$\begin{array}{cccc} & & & & & \\ & & & & \\ \text{State} & = & \text{fetch} \\ \text{IR} & = & \text{A} \leftarrow \text{MEM}[4] \\ \text{A} & = & 0 \\ \end{array}$	Address 0 1 2 3 4 5 6 7	Value 0 0 0 0 10 20 0 0	Address 0 1 2 3 4 5 6 7	TAPE Semantics A ← MEM[4] A ← A + MEM[5] MEM[6] ← A halt nop nop nop nop

Notes:



Example (compute 10 + 20)					
		ИЕМ] [TAPE
	Address	Value		Address	Semantics
	0	0	1 [0	$A \leftarrow MEM[4]$
CPU	1	0		1	$A \leftarrow A + MEM[5]$
state = execute	2	0		2	$MEM[6] \leftarrow A$
$IR = A \leftarrow MEM[4]$	3	0		3	halt
A = 10	4	10		4	nop
	5	20		5	nop
	6	0		6	nop
	7	0		7	nop

© Daniel Page (catephristol.ac.u.)

Computer Architecture

© Winversity of Sit # c144

BRISTOL git # c144

Part 1: ASCC: a Harvard architecture (4)

Example (compute 10 + 20)					
		ИEМ			TAPE
	Address	Value		Address	Semantics
	0	0	1	0	$A \leftarrow MEM[4]$
CPU	1	0		1	$A \leftarrow A + MEM[5]$
state = fetch	2	0		2	$MEM[6] \leftarrow A$
$IR = A \leftarrow A + MEM[5]$	3	0		3	halt
A = 10	4	10		4	nop
	5	20		5	nop
	6	0		6	nop
	7	0		7	nop
			_		1

Notes:	



Example (compute 10 + 20) MEM TAPE Address Value Address Semantics $\mathsf{A} \leftarrow \mathsf{MEM}[4]$ 0 CPU $A \leftarrow A + MEM[5]$ $MEM[6] \leftarrow A$ 1 0 1 execute 2 0 2 state IR $= A \leftarrow A + MEM[5]$ 3 3 halt = 30 10 nop 5 5 20 nop 6 7 nop nop 0 6 0

© Daniel Page ("suspin: istol..ac.ul")

Computer Architecture

© University of BRISTOL

git # c144985a @ 2024-10-14

Part 1: ASCC: a Harvard architecture (4)

Example (compute 10 + 20)				
$\begin{array}{ccc} & & & & \\ & & & & \\ \text{State} & = & \text{fetch} \\ \text{IR} & = & \text{MEM[6]} \leftarrow \text{A} \\ \text{A} & = & 30 \end{array}$	Address 0 1 2 3 4 5 6 7	Value 0 0 0 0 0 10 20 0 0	Address 0 1 2 3 4 5 6 7	TAPE

Notes:

Notes:

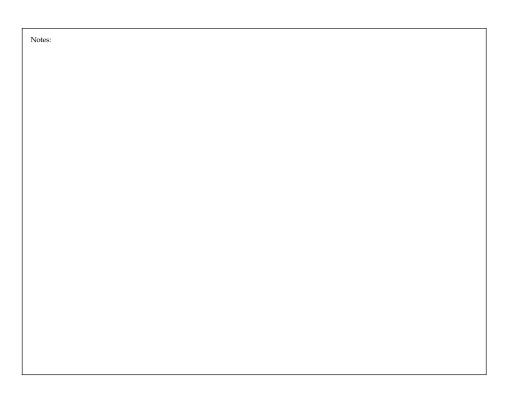


xample (compute 10 + 20)				
	N	ИEМ		TAPE
	Address	Value	Address	Semantics
	0	0	0	$A \leftarrow MEM[4]$
CPU	1	0	1	$A \leftarrow A + MEM[5]$
state = execute	2	0	2	$MEM[6] \leftarrow A$
$IR = MEM[6] \leftarrow A$	3	0	3	halt
A = 30	4	10	4	nop
		20	5	nop
	6	30	6	nop
	7	0	7	nop

Part 1: ASCC: a Harvard architecture (4)

Example (compute 10 + 20)					
	N	ИEМ	1		TAPE
	Address	Value	Ac	dress	Semantics
	0	0		0	$A \leftarrow MEM[4]$
CPU	1	0		1	$A \leftarrow A + MEM[5]$
state = fetch	2	0		2	MEM[6] ← A
IR = halt	3	0		3	halt
A = 30	4	10		4	nop
	5	20		5	nop
	6	30		6	nop
	7	0		7	nop





Example (compute 10 + 20)

		CPU	
state	=	execute	
IR	=	halt	
Α	=	30	

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

TAPE					
Address	Semantics				
0	$A \leftarrow MEM[4]$				
1	$A \leftarrow A + MEM[5]$				
2	$MEM[6] \leftarrow A$				
3	halt				
4	nop				
5	nop				
6	nop				
7	nop				

© Daniel Page (csdsp@bristol.ac.uk)

Computer Architecture



git # c144985a @ 2024-10-14

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.,

$$\begin{array}{ccccc} \text{NOP} & \mapsto & \\ \text{MOV x} & \mapsto & \text{A} \leftarrow \text{x} \\ \text{MOV y} & \mapsto & \text{A} \leftarrow \text{MEM[y]} \\ \text{MOV y} & \mapsto & \text{MEM[y]} \leftarrow \text{A} \\ \text{ADDC y} & \mapsto & \text{A} \leftarrow \text{A} + \text{MEM[y]} + \text{carry} \\ \text{SUBB y} & \mapsto & \text{A} \leftarrow \text{A} - \text{MEM[y]} - \text{carry} \\ \text{XRL y} & \mapsto & \text{A} \leftarrow \text{A} \oplus \text{MEM[y]} \end{array}$$

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:	

Notes:		



Hang on, what's the point of having one memory for data and another one for instructions ... they're the same!

If you keep real data and data that represents instructions both in one memory, you get a different stored-program architecture [14].



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

University of BRISTOL

University of BRISTOL

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,
 - ▶ 45.0m² footprint; 7.8t weight,
 - ▶ 1000-element, 44-bit memory (i.e., binary representation),
 - ▶ upto ~ 1160 operations per-second,
 - programs read from magnetic tape into memory via a tape reader.



- 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
- Zuse described the same concept, circa 1936, in a patent application for his range of early computers,
 von Neumann worked on the EDVAC project with Mauchly and Eckert, circa 1945, who previously built the ENIAC, and
 Turing described a stored-program computer, the Piol ACE, circa 1946.

Notes:			

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



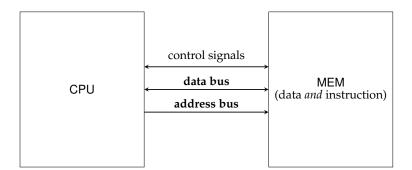




Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (2)

- **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:	
Notes:	

Notes:			

Example (instruction set) • 00nnnn means nop. • 10nnnn means halt. • 20nnnn means $A \leftarrow n$. • 21nnnn means $MEM[n] \leftarrow A$. • 22nnnn means $A \leftarrow MEM[n]$. • 30nnnn means $A \leftarrow A + MEM[n]$. • 31nnnn means $A \leftarrow A - MEM[n]$. • 32nnnn means $A \leftarrow A - MEM[n]$. • 40nnnn means $A \leftarrow A \oplus MEM[n]$. • 40nnnn means $A \leftarrow A \oplus MEM[n]$. • 41nnnn means $A \leftarrow A \oplus MEM[n]$. • 41nnnn means $A \leftarrow A \oplus MEM[n]$. • 41nnnn means $A \leftarrow A \oplus MEM[n]$.

© Daniel Page (catephristol.ac.ul)

Computer Architecture

© Winiversity of git # c144985a @ 2024-10-14

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 = \bot (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:		
Notes:		
notes.		

Example (compute 10 + 20)								
					MEM			
					Address	Value	Semantics	
			CPU		0	220004	$A \leftarrow MEM[4]$	
	state	=	reset		1	300005	$A \leftarrow A + MEM[5]$	
	PC	=	0		2	210006	$MEM[6] \leftarrow A$	
	IR	=			3	100000	halt	
		=			4	10	nop	
	Α	=	0		5	20	nop	
					6	0	nop	
					7	0	nop	

© Daniel Page (categoristo). ac.ul)

Computer Architecture

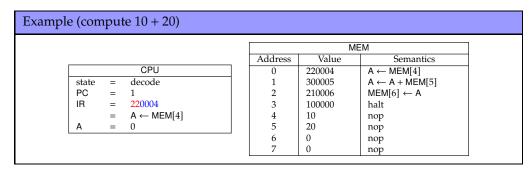
© University of University of BRISTOL git # c144985a @ 2024-10-14

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

$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$

Notes:	



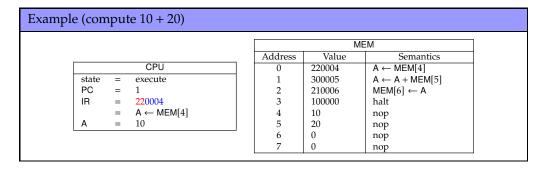


© Daniel Page (suspension Loc.uk)

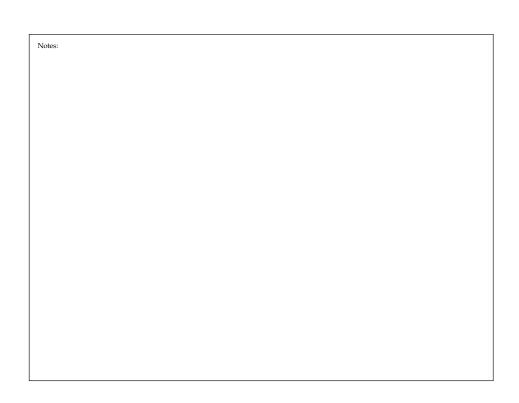
Computer Architecture

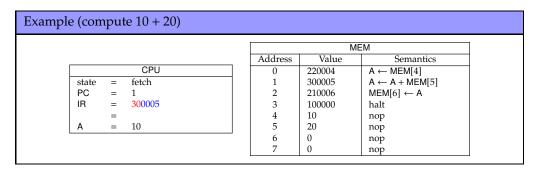
© Wilversity of BRISTOL git # c144985a @ 2024-10-14

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



Notes:	



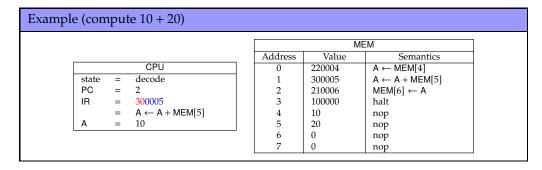


© Daniel Page (<u>categoristol.st.cat</u>)

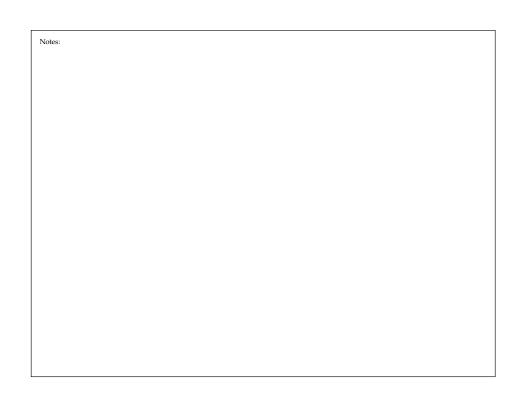
Computer Architecture

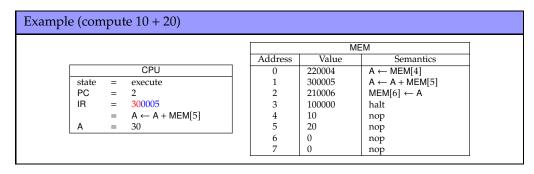
© Witwersity of BRISTOL git # c144985a @ 2024-10-14

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



Notes:		





© Daniel Page (<u>categoristol.st.cat</u>)

Computer Architecture

© Witwersity of BRISTOL git # c144985a @ 2024-10-14

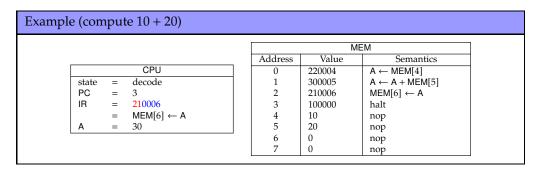
Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (5)

Example (compute 10 + 20)							
		М	EM				
	Address	Value	Semantics				
CPU	0	220004	$A \leftarrow MEM[4]$				
state = fetch	1	300005	$A \leftarrow A + MEM[5]$				
PC = 2	2	210006	MEM[6] ← A				
IR = 210006	3	100000	halt				
=	4	10	nop				
A = 30	5	20	nop				
	6	0	nop				
	7	0	nop				
	•						

Notes:	
1000	
<u> </u>	







© Daniel Page (suspin: stol.ac.ul)

Computer Architecture

© University of

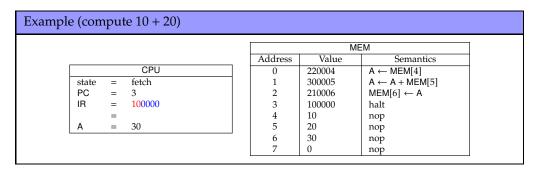
Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (5)

Example (compute 10 + 20)							
				Address	M Value	EM Semantics	
		CPU		Address 0	220004	Semantics $A \leftarrow MEM[4]$	
state	=	execute		1	300005	$A \leftarrow M \vdash M \vdash M[5]$	
PC	=	3		2	210006	MEM[6] ← A	
IR	=	210006		3	100000	halt	
	=	$MEM[6] \leftarrow A$		4	10	nop	
A	=	30		5	20	nop	
,		_		6	30	nop	
				7	0	nop	
						<u> </u>	

Notes:	

Notes:		



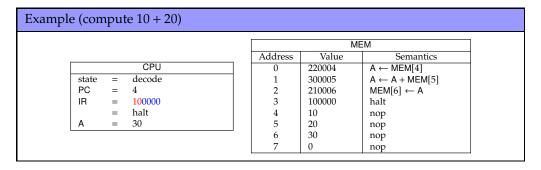


© Daniel Page (suspin: stol.ac.ul)

Computer Architecture

© University of

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



Notes:	
1000	
<u> </u>	



Examp	le (con	npu	te 10 + 20)				
						N	1EM
					Address	Value	Semantics
			CPU	1	0	220004	$A \leftarrow MEM[4]$
	state	=	execute	Ī	1	300005	$A \leftarrow A + MEM[5]$
	PC	=	4		2	210006	$MEM[6] \leftarrow A$
	IR	=	100000		3	100000	halt
		=	halt		4	10	nop
	Α	=	30		5	20	nop
					6	30	nop
					7	0	nop

© Daniel Page (<u>categoristol.st.cat</u>)

Computer Architecture

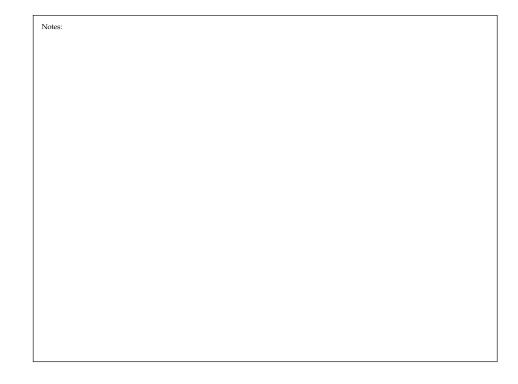
© Witwersity of BRISTOL git # c144985a @ 2024-10-14

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

Example (compute 10 + 20, including an infinite loop)					
		М	EM		
	Address	Value	Semantics		
CPU	0	220004	$A \leftarrow MEM[4]$		
state = reset	1	300005	$A \leftarrow A + MEM[5]$		
PC = 0	2	210006	$MEM[6] \leftarrow A$		
IR =	3	400000	PC ← 0		
=	4	10	nop		
A = 0	5	20	nop		
	6	0	nop		
	7	0	nop		

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

Notes:	·



Example (compute 10 + 20, including an infinite loop) Address Value Semantics 220004 $A \leftarrow MEM[4]$ state fetch 1 300005 $A \leftarrow A + MEM[5]$ = 2 210006 $MEM[6] \leftarrow A$ 0 220004 3 4 5 400000 $PC \leftarrow 0$ 10 nop 0 20 nop 6 0 nop 0 nop

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

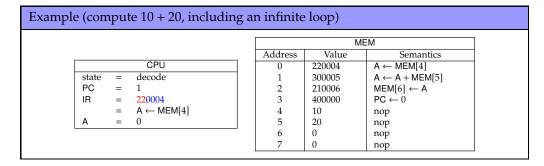
© Daniel Page (<u>subspirate to Luc.us</u>)

Computer Architecture

BRISTOL

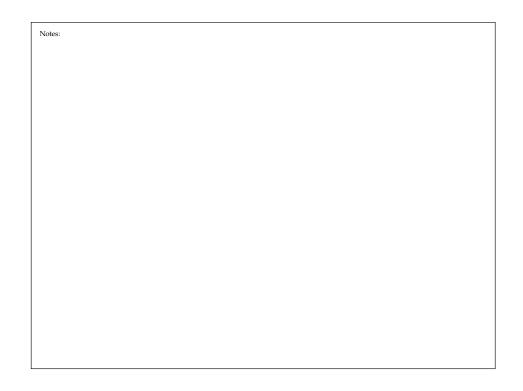
git # c144985a @ 2024-10-14

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



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

Notes:		



Example (compute 10 + 20, including an infinite loop) Address Value Semantics CPU 220004 $A \leftarrow MEM[4]$ state execute 1 300005 $A \leftarrow A + MEM[5]$ = 2 210006 $MEM[6] \leftarrow A$ 220004 3 4 5 400000 $PC \leftarrow 0$ $= A \leftarrow MEM[4]$ 10 nop 10 20 nop 6 0 nop 0 nop

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

© Daniel Page (<u>categorate to Luc.us</u>)

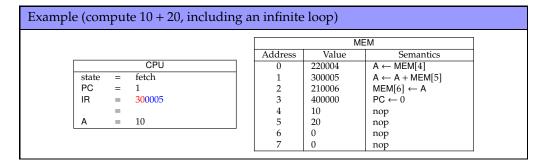
Computer Architecture

Size University of

BRISTOL

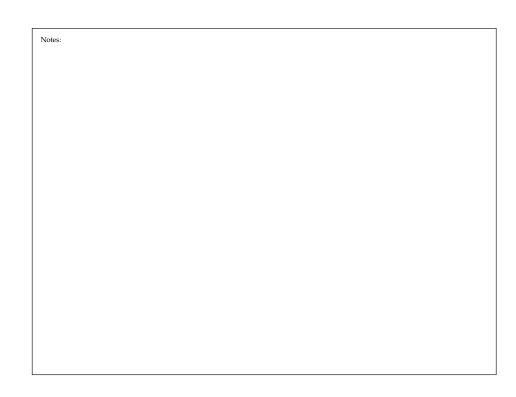
git # c144985a @ 2024-10-14

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



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

Notes:		

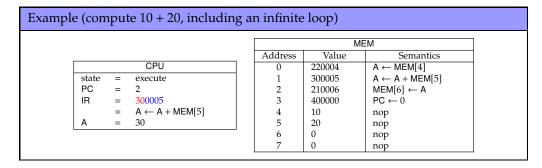


Example (compute 10 + 20, including an infinite loop) Address Value Semantics CPU 220004 $A \leftarrow MEM[4]$ state decode 1 300005 $A \leftarrow A + MEM[5]$ = 2 $MEM[6] \leftarrow A$ 2 210006 300005 3 4 5 400000 $PC \leftarrow 0$ $A \leftarrow A + MEM[5]$ 10 nop 10 20 nop 6 0 nop 0 nop

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

JL git # c144985a @ 202

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



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

Notes:	
	_



Example (compute 10 + 20, including an infinite loop) Address Value Semantics 220004 $A \leftarrow MEM[4]$ state fetch 1 300005 $A \leftarrow A + MEM[5]$ = 2 210006 $MEM[6] \leftarrow A$ 2 = 210006 3 4 5 400000 $PC \leftarrow 0$ 10 nop 30 20 nop 6 0 nop 0 nop

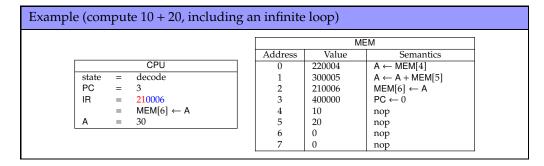
stored program
$$\Rightarrow$$
 implication #1 =
$$\begin{cases} 1. \text{ we can control PC, } so \\ 2. \text{ we can, e.g., write loops.} \end{cases}$$

© Daniel Page ("sdepthristoliac.ut")

Computer Architecture

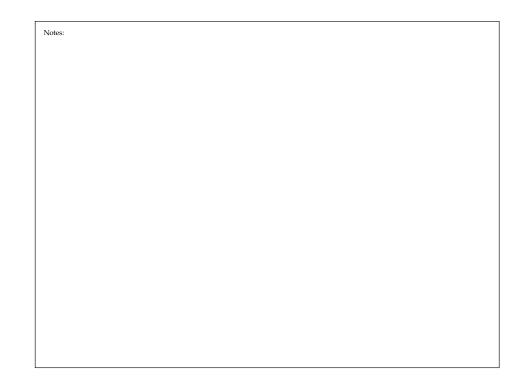
© University of University of BRISTOL git # c144985a @ 2024-10-14

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



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

Notes:	



Example (compute 10 + 20, including an infinite loop) Address Value Semantics CPU 220004 $A \leftarrow MEM[4]$ state execute 1 300005 $A \leftarrow A + MEM[5]$ = 2 210006 $MEM[6] \leftarrow A$ 3 = 210006 3 4 5 400000 $PC \leftarrow 0$ $MEM[6] \leftarrow A$ 10 nop 30 20 nop 6 30 nop 0 nop

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

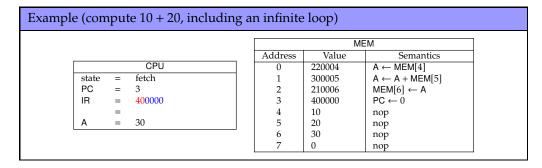
© Daniel Page (<u>categoristol.ac.u.</u>)

Computer Architecture

© Daniel Page (<u>categoristol.ac.u.</u>)

Eliminersity of BRISTOL git # c144985a @

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



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

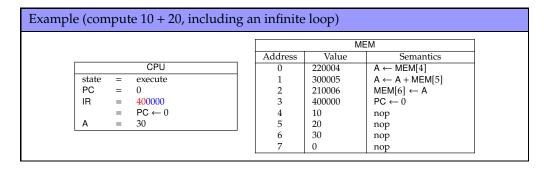
Notes:		



Example (compute 10 + 20, including an infinite loop) Address Value Semantics 220004 $A \leftarrow MEM[4]$ state decode 1 300005 $A \leftarrow A + MEM[5]$ = 2 210006 $MEM[6] \leftarrow A$ 400000 3 4 5 400000 $PC \leftarrow 0$ $PC \leftarrow 0$ 10 nop 30 20 nop 6 30 nop 0 nop

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

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



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

Notes:		



Example (compute 10 + 20, including an infinite loop) Address Value Semantics 220004 $A \leftarrow MEM[4]$ state fetch 1 300005 $A \leftarrow A + MEM[5]$ = 2 210006 $MEM[6] \leftarrow A$ 0 220004 3 4 5 400000 $PC \leftarrow 0$ 10 nop 30 20 nop 6 30 nop 0 nop

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

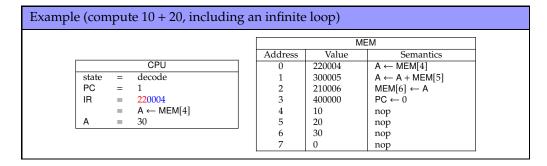
```
© Daniel Page (<u>adaptivistol.ac.u.</u>)

Computer Architecture

© Daniel Page (<u>adaptivistol.ac.u.</u>)

Ele University of BRISTOL git # c144985a @ 2024-10-14
```

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



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

Notes:	



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

		CPU
state	=	execute
PC	=	1
IR	=	220004
	=	$A \leftarrow MEM[4]$
Α	=	10

	ME	EM
Address	Value	Semantics
0	220004	$A \leftarrow MEM[4]$
1	300005	$A \leftarrow A + MEM[5]$
2	210006	$MEM[6] \leftarrow A$
3	400000	PC ← 0
4	10	nop
5	20	nop
6	30	nop
7	0	nop

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

© Daniel Page (csdsp@bristol.ac.uk)

Computer Architecture

University of BRISTOL

git # c144985a @ 2024-10-

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

ſ			CPU
ſ	state	=	reset
	PC	=	0
	IR	=	
		=	
	Α	=	0

	ME	EM
Address	Value	Semantics
0	220004	$A \leftarrow MEM[4]$
1	300005	$A \leftarrow A + MEM[5]$
2	210003	$MEM[3] \leftarrow A$
3	100000	halt
4	10	nop
5	20	nop
6	0	nop
7	0	nop

stored program
$$\Rightarrow$$
 implication #2 =
$$\begin{cases} 1. \text{ data } \text{and } \text{instructions are stored in MEM, so} \\ 2. \text{ we can, e.g., write } \text{self-modifying } \text{code.} \end{cases}$$

Notes:		

Notes:			

Example (compute 10 + 20, including programming mistake)

		CPU
state	=	fetch
PC	=	0
IR	=	220004
	=	
Α	=	0

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

stored program
$$\Rightarrow$$
 implication #2 =
$$\begin{cases} 1. \text{ data } and \text{ instructions are stored in MEM, } so \\ 2. \text{ we can, e.g., write } self-modifying \text{ code.} \end{cases}$$

© Daniel Page (csdsp@bristol.ac.uk)

Computer Architecture

University of BRISTOL

git # c144985a @ 2024-10-1

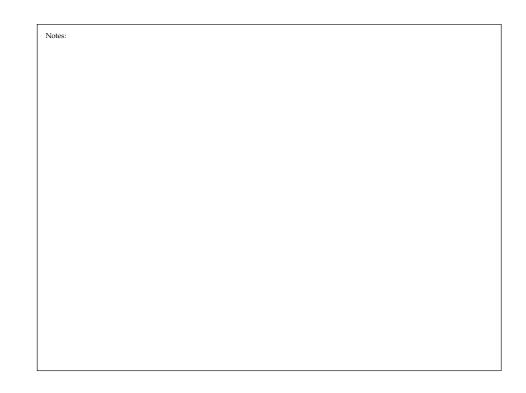
Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (7)

		CPU
state	=	decode
PC	=	1
IR	=	220004
	=	$A \leftarrow MEM[4]$
Α	=	0

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

stored program
$$\Rightarrow$$
 implication #2 = $\begin{cases} 1. \text{ data } and \text{ instructions are stored in MEM, } so \\ 2. \text{ we can, e.g., write } self-modifying code. \end{cases}$

	Notes:
_	



Example (compute 10 + 20, including programming mistake)

	CPU			
state	=	execute		
PC	=	1		
IR	=	220004		
	=	$A \leftarrow MEM[4]$		
Α	=	10		

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

stored program
$$\Rightarrow$$
 implication #2 =
$$\begin{cases} 1. \text{ data } and \text{ instructions are stored in MEM, } so \\ 2. \text{ we can, e.g., write } self-modifying \text{ code.} \end{cases}$$

© Daniel Page (csdsp@bristol.ac.uk Computer Architecture University of BRISTOL

git # c144985a @ 2024-10-1

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

		CPU
state	=	fetch
PC	=	1
IR	=	300005
	=	
Α	=	10

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

stored program
$$\Rightarrow$$
 implication #2 =
$$\begin{cases} 1. \text{ data } \text{and } \text{instructions are stored in MEM, so} \\ 2. \text{ we can, e.g., write } \text{self-modifying } \text{code.} \end{cases}$$

Notes:	
Notes.	



Example (compute 10 + 20, including programming mistake)

		CPU
state	=	decode
PC	=	2
IR	=	300005
	=	$A \leftarrow A + MEM[5]$
Α	=	10

MEM				
Value	Semantics			
220004	$A \leftarrow MEM[4]$			
300005	$A \leftarrow A + MEM[5]$			
210003	MEM[3] ← A			
100000	halt			
10	nop			
20	nop			
0	nop			
0	nop			
	Value 220004 300005 210003 100000 10			

stored program
$$\Rightarrow$$
 implication #2 = $\begin{cases} 1. \text{ data } and \text{ instructions are stored in MEM, } so \\ 2. \text{ we can, e.g., write } self-modifying \text{ code.} \end{cases}$

© Daniel Page (csdsp@bristol.ac.uk)

Computer Architecture

University of BRISTOL

git # c144985a @ 2024-10-

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

Example (compute 10 + 20, including programming mistake)

		CPU
state	=	execute
PC	=	2
IR	=	300005
	=	$A \leftarrow A + MEM[5]$
Α	=	30

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

stored program
$$\Rightarrow$$
 implication #2 =
$$\begin{cases} 1. \text{ data } and \text{ instructions are stored in MEM, } so \\ 2. \text{ we can, e.g., write } self-modifying \text{ code.} \end{cases}$$

University of BRISTOL

Notes:		



$Example \ (compute \ 10 + 20, including \ programming \ mistake)$

		CPU
state	=	fetch
PC	=	2
IR	=	210003
	=	
Α	=	30
		-

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

stored program
$$\Rightarrow$$
 implication #2 =
$$\begin{cases} 1. \text{ data } and \text{ instructions are stored in MEM, } so \\ 2. \text{ we can, e.g., write } self-modifying \text{ code.} \end{cases}$$

© Daniel Page (csdsp@bristol.ac.uk)

Computer Architecture

University of BRISTOL

git # c144985a @ 2024-10-1

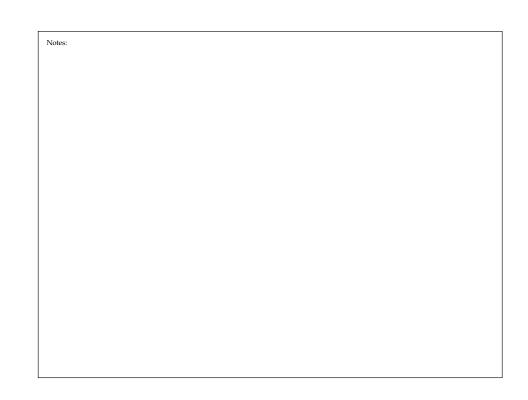
Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (7)

		CPU
state	=	decode
PC	=	3
IR	=	210003
	=	$MEM[3] \leftarrow A$
Α	=	30

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

$$\begin{array}{ll} \text{stored} \\ \text{program} & \Rightarrow \text{ implication #2} \\ \end{array} = \left\{ \begin{array}{ll} 1. \text{ data and instructions are stored in MEM, so} \\ 2. \text{ we can, e.g., write } \textit{self-modifying code.} \end{array} \right.$$

Notes:		



Example (compute 10 + 20, including programming mistake)

		CPU
state	=	execute
PC	=	3
IR	=	210003
	=	$MEM[3] \leftarrow A$
Α	=	30

MEM				
Address	Value	Semantics		
0	220004	$A \leftarrow MEM[4]$		
1	300005	$A \leftarrow A + MEM[5]$		
2	210003	$MEM[3] \leftarrow A$		
3	30	nop		
4	10	nop		
5	20	nop		
6	0	nop		
7	0	nop		

stored program
$$\Rightarrow$$
 implication #2 =
$$\begin{cases} 1. \text{ data } and \text{ instructions are stored in MEM, } so \\ 2. \text{ we can, e.g., write } self-modifying \text{ code.} \end{cases}$$

© Daniel Page (csdsp@bristol.ac.uk)

Computer Architecture

University of BRISTOL

git # c144985a @ 2024-10-1

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

		CPU
state	=	fetch
PC	=	3
IR	=	000030
	=	
Α	=	30

	MEM				
Address	Value	Semantics			
0	220004	$A \leftarrow MEM[4]$			
1	300005	$A \leftarrow A + MEM[5]$			
2	210003	$MEM[3] \leftarrow A$			
3	30	nop			
4	10	nop			
5	20	nop			
6	0	nop			
7	0	nop			

$$\begin{array}{ll} \text{stored} \\ \text{program} & \Rightarrow \text{ implication #2} \\ \end{array} = \left\{ \begin{array}{ll} 1. \text{ data and instructions are stored in MEM, so} \\ 2. \text{ we can, e.g., write } \textit{self-modifying code.} \end{array} \right.$$







Example (compute 10 + 20, including programming mistake)

		CPU	
state	=	decode	
PC	=	4	
IR	=	000030	
	=	nop	
Α	=	nop 30	

MEM				
Address	Value	Semantics		
0	220004	$A \leftarrow MEM[4]$		
1	300005	$A \leftarrow A + MEM[5]$		
2	210003	$MEM[3] \leftarrow A$		
3	30	nop		
4	10	nop		
5	20	nop		
6	0	nop		
7	0	nop		

stored program
$$\Rightarrow$$
 implication #2 =
$$\begin{cases} 1. \text{ data } and \text{ instructions are stored in MEM, } so \\ 2. \text{ we can, e.g., write } self-modifying \text{ code.} \end{cases}$$

© Daniel Page (csdsp@bristol.ac.uk)

Computer Architecture

University of BRISTOL

git # c144985a @ 2024-10-1

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	
Α	=	30	

	MEM					
Address	Value	Semantics				
0	220004	$A \leftarrow MEM[4]$				
1	300005	$A \leftarrow A + MEM[5]$				
2	210003	$MEM[3] \leftarrow A$				
3	30	nop				
4	10	nop				
5	20	nop				
6	0	nop				
7	0	nop				

$$\begin{array}{ll} \text{stored} \\ \text{program} & \Rightarrow \text{ implication #2} \\ \end{array} = \left\{ \begin{array}{ll} 1. \text{ data and instructions are stored in MEM, so} \\ 2. \text{ we can, e.g., write } \textit{self-modifying code.} \end{array} \right.$$

Notes:



$Example \ (compute \ 10 + 20, including \ programming \ mistake)$

		CPU
state	=	fetch
PC	=	4
IR	=	000010
	=	
Α	=	30

MEM				
Address	Value	Semantics		
0	220004	$A \leftarrow MEM[4]$		
1	300005	$A \leftarrow A + MEM[5]$		
2	210003	$MEM[3] \leftarrow A$		
3	30	nop		
4	10	nop		
5	20	nop		
6	0	nop		
7	0	nop		

stored program
$$\Rightarrow$$
 implication #2 =
$$\begin{cases} 1. \text{ data } and \text{ instructions are stored in MEM, } so \\ 2. \text{ we can, e.g., write } self-modifying \text{ code.} \end{cases}$$

© Daniel Page (csdsp@bristol.ac.uk)

Computer Architecture

University of BRISTOL

git # c144985a @ 2024-10-1

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

		CPU
state	=	decode
PC	=	5
IR	=	000010
	=	nop
Α	=	30

	MEM					
Address Value		Semantics				
0	220004	$A \leftarrow MEM[4]$				
1	300005	$A \leftarrow A + MEM[5]$				
2	210003	$MEM[3] \leftarrow A$				
3	30	nop				
4	10	nop				
5	20	nop				
6	0	nop				
7 0		nop				

stored program
$$\Rightarrow$$
 implication #2 =
$$\begin{cases} 1. \text{ data } \text{and } \text{instructions are stored in MEM, so} \\ 2. \text{ we can, e.g., write } \text{self-modifying } \text{code.} \end{cases}$$

Notes:	



Example (compute 10 + 20, including programming mistake) Address Value Semantics CPU 220004 $A \leftarrow MEM[4]$ execute 1 300005 $A \leftarrow A + MEM[5]$ = 2 210003 $MEM[3] \leftarrow A$ 5 000010 3 4 5 30 nop 10 nop 30 20 nop 6 0 nop nop

$$\begin{array}{ll} \text{stored} \\ \text{program} & \Rightarrow \text{ implication #2} \\ \end{array} = \left\{ \begin{array}{ll} 1. \text{ data } \textit{and} \text{ instructions are stored in MEM, } \textit{so} \\ 2. \text{ we can, e.g., write } \textit{self-modifying } \text{code.} \end{array} \right.$$

© Daniel Page (csdsp@bristol.ac.uk)	University of BRISTOL	
Computer Architecture	S BRISTOL	git # c144985a @ 2024-10-14

Part 2: EDVAC: a Princeton (aka. von Neumann) architecture (8)

- **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:		
Notes:		
Notes:		
Notes:		



Notes:

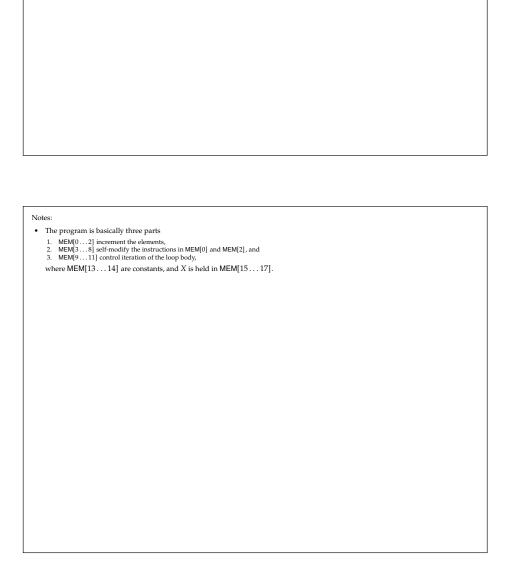
- **Beware**: self-modifying code isn't *just* a destructive "problem".
- ► Solution: use self-modifying code constructively.

 - ▶ 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].



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

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)					
MEM					
	Address	Value	Semantics	1	
	0	220015	A ← MEM[15]	1	
	1	300013	$A \leftarrow A + MEM[13]$		
	2	210015	$MEM[15] \leftarrow A$		
	3	220000	$A \leftarrow MEM[0]$		
	4	300013	$A \leftarrow A + MEM[13]$		
CPU	5	210000	$MEM[0] \leftarrow A$		
state = reset	6	220002	$A \leftarrow MEM[2]$		
PC = 0	7	300013	$A \leftarrow A + MEM[13]$		
IR =	8	210002	$MEM[2] \leftarrow A$		
=	9	220000	$A \leftarrow MEM[0]$		
A = 0	10	310014	$A \leftarrow A - MEM[14]$		
	11	420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$		
	12	100000	halt		
	13	1	nop		
	14	220018	$A \leftarrow MEM[18]$		
	15	0	nop		
	16	1	nop		
	17	2	nop		
				-	







Notes:

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[15]$ 220015 300013 $A \leftarrow A + MEM[13]$ 2 210015 $MEM[15] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = fetch 6 220002 $A \leftarrow MEM[2]$ PC 7 300013 $A \leftarrow A + MEM[13]$ 8 IR 210002 $MEM[2] \leftarrow A$ = 220015 9 220000 $A \leftarrow MEM[0]$ 10 310014 $A \leftarrow A - MEM[14]$ = 0 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 nop 14 220018 $A \leftarrow MEM[18]$ 15 nop 16 nop 17 nop



University of BRISTOL

git # c144985a @ 2024-10-1

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

Example (increment elements in the se	equence $X = \langle 0, 1, 2 \rangle$
	NAENA

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

MEM					
Address	Value	Semantics			
0	220015	$A \leftarrow MEM[15]$			
1	300013	$A \leftarrow A + MEM[13]$			
2	210015	$MEM[15] \leftarrow A$			
3	220000	$A \leftarrow MEM[0]$			
4	300013	$A \leftarrow A + MEM[13]$			
5	210000	$MEM[0] \leftarrow A$			
6	220002	$A \leftarrow MEM[2]$			
7	300013	$A \leftarrow A + MEM[13]$			
8	210002	$MEM[2] \leftarrow A$			
9	220000	$A \leftarrow MEM[0]$			
10	310014	$A \leftarrow A - MEM[14]$			
11	420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$			
12	100000	halt			
13	1	nop			
14	220018	$A \leftarrow MEM[18]$			
15	0	nop			
16	1	nop			
17	2	nop			
	•				





Notes:

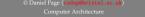
- · The program is basically three parts
- MEM[0...2] increment the elements,
- 2. MEM[3...8] self-modify the instructions in MEM[0] and MEM[2], and
- MEM[9...11] control iteration of the loop body,

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

Notes:

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

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[15]$ 220015 300013 $A \leftarrow A + MEM[13]$ 2 210015 $MEM[15] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state execute 6 220002 $A \leftarrow MEM[2]$ = PC 7 $A \leftarrow A + MEM[13]$ 300013 8 210002 $MEM[2] \leftarrow A$ 220015 9 $= A \leftarrow MEM[15]$ 220000 $A \leftarrow MEM[0]$ = 010 310014 $A \leftarrow A - MEM[14]$ 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 nop 17 nop



University of BRISTOL

git # c144985a @ 2024-10-1

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

Example (increment elements in the sec	querice A	- (0,
	Address	Valı

		CPU
state	=	fetch
PC	=	1
IR	=	300013
	=	
Α	=	0

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





Notes:

The program is basically three parts

MEM[0...2] increment the elements,

2. $\mathsf{MEM}[3\dots 8]$ self-modify the instructions in $\mathsf{MEM}[0]$ and $\mathsf{MEM}[2]$, and

MEM[9...11] control iteration of the loop body,

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

Notes:

· The program is basically three parts

MEM[0...2] increment the elements,

MEM[3...8] self-modify the instructions in MEM[0] and MEM[2], and

MEM[9...11] control iteration of the loop body,

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[15]$ 220015 300013 $A \leftarrow A + MEM[13]$ 2 210015 $MEM[15] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = decode 6 220002 $A \leftarrow MEM[2]$ PC 2 7 300013 $A \leftarrow A + MEM[13]$ 8 210002 $MEM[2] \leftarrow A$ 300013 $= A \leftarrow A + MEM[13]$ 9 220000 $A \leftarrow MEM[0]$ 0 10 310014 $A \leftarrow A - MEM[14]$ = 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 nop 16 nop 17 nop



University of BRISTOL

git # c144985a @ 2024-10-1

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

Examp!	le ((increment e	lements ii	n the s	sequence ?	$X = \langle 0 \rangle$	(0,1,2)
			•		•		

 $\begin{array}{ccc} & & & & & \\ \text{State} & = & \text{execute} \\ \text{PC} & = & 2 \\ \text{IR} & = & 300013 \\ & = & \text{A} \leftarrow \text{A} + \text{MEM}[13] \\ \text{A} & = & 1 \end{array}$

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





Notes:

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

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

Notes:

- · The program is basically three parts
- MEM[0...2] increment the elements,
- MEM[0...2] Interinct the centerto,
 MEM[0] and MEM[0] and MEM[2], and
- MEM[9...11] control iteration of the loop body,

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[15]$ 220015 300013 $A \leftarrow A + MEM[13]$ 2 210015 $MEM[15] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = fetch 6 220002 $A \leftarrow MEM[2]$ PC 2 7 300013 $A \leftarrow A + MEM[13]$ 8 IR 210002 $MEM[2] \leftarrow A$ = 210015 9 220000 $A \leftarrow MEM[0]$ 10 310014 $A \leftarrow A - MEM[14]$ = 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 nop 14 220018 $A \leftarrow MEM[18]$ 15 nop 16 nop 17 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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

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

MEM				
Value	Semantics			
220015	$A \leftarrow MEM[15]$			
300013	$A \leftarrow A + MEM[13]$			
210015	$MEM[15] \leftarrow A$			
220000	$A \leftarrow MEM[0]$			
300013	$A \leftarrow A + MEM[13]$			
210000	$MEM[0] \leftarrow A$			
220002	$A \leftarrow MEM[2]$			
300013	$A \leftarrow A + MEM[13]$			
210002	$MEM[2] \leftarrow A$			
220000	$A \leftarrow MEM[0]$			
310014	$A \leftarrow A - MEM[14]$			
420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$			
100000	halt			
1	nop			
220018	$A \leftarrow MEM[18]$			
0	nop			
1	nop			
2	nop			
	Value 220015 300013 210015 220000 300013 210000 220002 300013 210002 220000 310014 420000 100000 1 220018 0 1			





Notes:

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

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

Notes:

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

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[15]$ 220015 300013 $A \leftarrow A + MEM[13]$ 2 210015 $MEM[15] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state execute 6 220002 $A \leftarrow MEM[2]$ = PC 7 300013 $A \leftarrow A + MEM[13]$ 3 8 210002 $MEM[2] \leftarrow A$ 210015 9 = $MEM[15] \leftarrow A$ 220000 $A \leftarrow MEM[0]$ 10 310014 $A \leftarrow A - MEM[14]$ = 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 nop 16 nop 17 nop



University of BRISTOL

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

Example (increment elements in the sequence X = (0, 1, 2))

		CPU	
state	=	fetch	
PC	=	3	
IR	=	220000	
	=		
Α	=	1	1

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





Notes:

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

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

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

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$) Value Address Semantics $A \leftarrow MEM[15]$ 220015 300013 $A \leftarrow A + MEM[13]$ 2 210015 $MEM[15] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = decode 6 220002 $A \leftarrow MEM[2]$ PC 7 $A \leftarrow A + MEM[13]$ 300013 8 $MEM[2] \leftarrow A$ 220000 210002 9 $= A \leftarrow MEM[0]$ 220000 $A \leftarrow MEM[0]$ 10 $A \leftarrow A - MEM[14]$ = 310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 nop 16 nop 17 nop



University of BRISTOL

git # c144985a @ 2024-10-1

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	

= 220000= $A \leftarrow MEM[0]$ = 220015

MEM Address Value Semantics					
Value	Semantics				
220015	$A \leftarrow MEM[15]$				
300013	$A \leftarrow A + MEM[13]$				
210015	$MEM[15] \leftarrow A$				
220000	$A \leftarrow MEM[0]$				
300013	$A \leftarrow A + MEM[13]$				
210000	$MEM[0] \leftarrow A$				
220002	$A \leftarrow MEM[2]$				
300013	$A \leftarrow A + MEM[13]$				
210002	$MEM[2] \leftarrow A$				
220000	$A \leftarrow MEM[0]$				
310014	$A \leftarrow A - MEM[14]$				
420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$				
100000	halt				
1	nop				
220018	$A \leftarrow MEM[18]$				
1	nop				
1	nop				
2	nop				
	Value 220015 300013 210015 220000 300013 210000 220002 300013 210002 220000 310014 420000 1000000 1 220018 1				





Notes:

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

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

The program is basically three parts

MEM[0...2] increment the elements,
MEM[9...1] 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 = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[15]$ 220015 300013 $A \leftarrow A + MEM[13]$ 2 210015 $MEM[15] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = fetch 6 220002 $A \leftarrow MEM[2]$ PC 7 300013 $A \leftarrow A + MEM[13]$ 8 210002 $MEM[2] \leftarrow A$ = 300013 9 220000 $A \leftarrow MEM[0]$ 220015 10 310014 $A \leftarrow A - MEM[14]$ = 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 nop 17 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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

]	Exampl	.e (increment el	lements i	in the	sequenc	e X =	$\langle 0, 1, 2 \rangle$)

		CPU
state	=	decode
PC	=	5
IR	=	300013
	=	$A \leftarrow A + MEM[13]$
Α	=	220015

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





Notes:

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

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

Notes:

- · The program is basically three parts
- MEM[0...2] increment the elements,
- MEM[0...2] Interinct the centerto,
 MEM[0] and MEM[0] and MEM[2], and
- MEM[9...11] control iteration of the loop body,

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$) Value Address Semantics $A \leftarrow MEM[15]$ 220015 300013 $A \leftarrow A + MEM[13]$ 2 210015 $MEM[15] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state execute 6 220002 $A \leftarrow MEM[2]$ = PC 5 7 $A \leftarrow A + MEM[13]$ 300013 8 $MEM[2] \leftarrow A$ 300013 210002 $A \leftarrow A + MEM[13]$ 9 220000 $A \leftarrow MEM[0]$ 220016 10 $A \leftarrow A - MEM[14]$ 310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 nop 17 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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

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

		CDII	
		CPU	
state	=	CPU fetch	
state PC	= =		
		fetch	
PC	=	fetch 5	

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





Notes:

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

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

Notes:

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

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$) Value Address Semantics $A \leftarrow MEM[15]$ 220015 300013 $A \leftarrow A + MEM[13]$ 2 210015 $MEM[15] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state decode 6 220002 $A \leftarrow MEM[2]$ = PC 7 $A \leftarrow A + MEM[13]$ 300013 8 $MEM[2] \leftarrow A$ 210000 210002 9 = $MEM[0] \leftarrow A$ 220000 $A \leftarrow MEM[0]$ 220016 10 $A \leftarrow A - MEM[14]$ = 310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 nop 17 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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	$A \leftarrow A + MEM[13]$		
2	210015	MEM[15] ← A		
3	220000	$A \leftarrow MEM[0]$		
4	300013	$A \leftarrow A + MEM[13]$		
5	210000	$MEM[0] \leftarrow A$		
6	220002	$A \leftarrow MEM[2]$		
7	300013	$A \leftarrow A + MEM[13]$		
8	210002	MEM[2] ← A		
9	220000	$A \leftarrow MEM[0]$		
10	310014	$A \leftarrow A - MEM[14]$		
11	420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$		
12	100000	halt		
13	1	nop		
14	220018	$A \leftarrow MEM[18]$		
15	1	nop		
16	1	nop		
17	2	nop		





Notes:

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

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

Notes:

The program is basically three parts

MEM[0...2] increment the elements,

MEM[3...8] self-modify the instructions in MEM[0] and MEM[2], and

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 = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[16]$ 220016 300013 $A \leftarrow A + MEM[13]$ 2 210015 $MEM[15] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = fetch 6 220002 $A \leftarrow MEM[2]$ PC 7 $A \leftarrow A + MEM[13]$ 300013 8 210002 $MEM[2] \leftarrow A$ 220002 9 220000 $A \leftarrow MEM[0]$ 220016 10 $A \leftarrow A - MEM[14]$ = 310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 nop 16 nop 17 nop



University of BRISTOL

git # c144985a @ 2024-10-1

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
	=	$A \leftarrow MEM[2]$
Α	=	220016

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





Notes:

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

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

Notes:

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

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics 220016 $A \leftarrow MEM[16]$ 300013 $A \leftarrow A + MEM[13]$ 2 210015 $MEM[15] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state execute 6 220002 $A \leftarrow MEM[2]$ = PC 7 $A \leftarrow A + MEM[13]$ 300013 8 $MEM[2] \leftarrow A$ 220002 210002 9 $A \leftarrow MEM[2]$ 220000 $A \leftarrow MEM[0]$ 210015 10 $A \leftarrow A - MEM[14]$ = 310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 nop 16 nop 17 nop



University of BRISTOL

git # c144985a @ 2024-10-1

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	

= 210015

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





Notes:

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

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

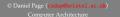
The program is basically three parts

MEM[0...2] increment the elements,

MEM[3...8] self-modify the instructions in MEM[0] and MEM[2], and

MEM[9...11] control iteration of the loop body,

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[16]$ 220016 300013 $A \leftarrow A + MEM[13]$ 2 210015 $MEM[15] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = decode 6 220002 $A \leftarrow MEM[2]$ PC 7 300013 $A \leftarrow A + MEM[13]$ 8 210002 $MEM[2] \leftarrow A$ 300013 $A \leftarrow A + MEM[13]$ 9 220000 $A \leftarrow MEM[0]$ 210015 10 310014 $A \leftarrow A - MEM[14]$ 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 nop 16 nop 17 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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

Example	(increment	elements	in the sec	quence λ	$X = \langle 0, 1,$.2>)

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





Notes:

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

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

Notes

- · The program is basically three parts
- MEM[0...2] increment the elements,
- MEM[0...2] Interinct the centerto,
 MEM[0] and MEM[0] and MEM[2], and
- MEM[9...11] control iteration of the loop body,

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[16]$ 220016 300013 $A \leftarrow A + MEM[13]$ 2 210015 $MEM[15] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = fetch 6 220002 $A \leftarrow MEM[2]$ PC 7 300013 $A \leftarrow A + MEM[13]$ 8 210002 $MEM[2] \leftarrow A$ = 210002 9 220000 $A \leftarrow MEM[0]$ 210016 10 310014 $A \leftarrow A - MEM[14]$ = 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 nop 17 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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

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

 CPU

 state
 =
 decode

 PC
 =
 9

 IR
 =
 210002

 =
 MEM[2] ← A

 A
 =
 210016

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





Notes:

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

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

Notes:

- · The program is basically three parts
- MEM[0...2] increment the elements,
- MEM[0...2] Interinct the centerto,
 MEM[0] and MEM[0] and MEM[2], and
- MEM[9...11] control iteration of the loop body,

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[16]$ 220016 300013 $A \leftarrow A + MEM[13]$ 2 210016 $MEM[16] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state execute 6 220002 $A \leftarrow MEM[2]$ = PC 7 $A \leftarrow A + MEM[13]$ 300013 8 210002 $MEM[2] \leftarrow A$ 210002 9 = $MEM[2] \leftarrow A$ 220000 $A \leftarrow MEM[0]$ 210016 10 $A \leftarrow A - MEM[14]$ = 310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 nop 14 220018 $A \leftarrow MEM[18]$ 15 nop 16 nop 17 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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

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

CPU				
CPU				
state = fetch		CPU fetch]

		CPU
state	=	fetch
PC	=	9
IR	=	220000
	=	
Α	=	210016

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





Notes:

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

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

Notes:

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

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[16]$ 220016 300013 $A \leftarrow A + MEM[13]$ 2 210016 $MEM[16] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = decode 6 220002 $A \leftarrow MEM[2]$ PC 10 7 300013 $A \leftarrow A + MEM[13]$ 8 210002 $MEM[2] \leftarrow A$ 220000 9 $A \leftarrow MEM[0]$ 220000 $A \leftarrow MEM[0]$ 210016 10 310014 $A \leftarrow A - MEM[14]$ = 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 nop 17 nop





git # c144985a @ 2024-10-1

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

Example	(increment e	elements ii	n the sec	quence 2	$X = \langle 0,$	1,2

CPU		
state	=	execute
PC	=	10
IR	=	220000
	=	$A \leftarrow MEM[0]$
Α	=	220016

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





Notes:

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

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

Notes:

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

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[16]$ 220016 300013 $A \leftarrow A + MEM[13]$ 2 210016 $MEM[16] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = fetch 6 220002 $A \leftarrow MEM[2]$ PC 10 7 300013 $A \leftarrow A + MEM[13]$ 8 210002 $MEM[2] \leftarrow A$ 310014 9 220000 $A \leftarrow MEM[0]$ 220016 10 310014 $A \leftarrow A - MEM[14]$ 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 nop 17 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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
	=	$A \leftarrow A - MEM[14]$
Α	=	220016

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





Notes:

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

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

Notes:

- · The program is basically three parts
- MEM[0...2] increment the elements,
- MEM[0...2] Interinct the centerto,
 MEM[0] and MEM[0] and MEM[2], and
- MEM[9...11] control iteration of the loop body,

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$) Value Address Semantics $A \leftarrow MEM[16]$ 220016 300013 $A \leftarrow A + MEM[13]$ 2 210016 $MEM[16] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state execute 6 220002 $A \leftarrow MEM[2]$ = PC 7 $A \leftarrow A + MEM[13]$ 11 300013 8 210002 $MEM[2] \leftarrow A$ 310014 $A \leftarrow A - MEM[14]$ 9 220000 $A \leftarrow MEM[0]$ 10 $A \leftarrow A - MEM[14]$ -2310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 nop 17 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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	$A \leftarrow MEM[16]$	
1	300013	$A \leftarrow A + MEM[13]$	
2	210016	MEM[16] ← A	
3	220000	$A \leftarrow MEM[0]$	
4	300013	$A \leftarrow A + MEM[13]$	
5	210000	$MEM[0] \leftarrow A$	
6	220002	$A \leftarrow MEM[2]$	
7	300013	$A \leftarrow A + MEM[13]$	
8	210002	$MEM[2] \leftarrow A$	
9	220000	$A \leftarrow MEM[0]$	
10	310014	$A \leftarrow A - MEM[14]$	
11	420000	$PC \leftarrow 0 \text{ iff. A} \neq 0$	
12	100000	halt	
13	1	nop	
14	220018	$A \leftarrow MEM[18]$	
15	1	nop	
16	1	nop	
17	2	nop	





Notes:

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

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

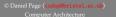
Notes:

- · The program is basically three parts
- MEM[0...2] increment the elements,
- MEM[0...2] Interinct the centerto,
 MEM[0] and MEM[0] and MEM[2], and
- MEM[9...11] control iteration of the loop body,

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

		CPU
		1 1
state	=	decode
PC	_	12
10	_	12
IR	=	420000
	_	$PC \leftarrow 0 \text{ iff. } A \neq 0$
	_	$FU \leftarrow 0 \text{ III. } A \neq 0$
Α	=	-2

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



University of BRISTOL

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
	=	$PC \leftarrow 0 \text{ iff. } A \neq 0$
Α	=	-2

Address	Value	
	varac	Semantics
0	220016	$A \leftarrow MEM[16]$
1	300013	$A \leftarrow A + MEM[13]$
2	210016	$MEM[16] \leftarrow A$
3	220000	$A \leftarrow MEM[0]$
4	300013	$A \leftarrow A + MEM[13]$
5	210000	$MEM[0] \leftarrow A$
6	220002	$A \leftarrow MEM[2]$
7	300013	$A \leftarrow A + MEM[13]$
8	210002	$MEM[2] \leftarrow A$
9	220000	$A \leftarrow MEM[0]$
10	310014	$A \leftarrow A - MEM[14]$
11	420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$
12	100000	halt
13	1	nop
14	220018	$A \leftarrow MEM[18]$
15	1	nop
16	1	nop
17	2	nop





Notes:

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

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

- 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,

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$) Value Address Semantics $A \leftarrow MEM[16]$ 220016 300013 $A \leftarrow A + MEM[13]$ 2 210016 $MEM[16] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = fetch 6 220002 $A \leftarrow MEM[2]$ PC 7 $A \leftarrow A + MEM[13]$ 300013 8 210002 $MEM[2] \leftarrow A$ = 220016 9 220000 $A \leftarrow MEM[0]$ 10 $A \leftarrow A - MEM[14]$ = -2310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 nop 14 220018 $A \leftarrow MEM[18]$ 15 nop 16 nop 17 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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

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

 $\begin{array}{rcl} & & & & & \\ \text{State} & = & \text{decode} \\ \text{PC} & = & 1 \\ \text{IR} & = & 220016 \\ & = & \text{A} \leftarrow \text{MEM}[16] \\ \text{A} & = & -2 \end{array}$

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





Notes:

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

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

Notes:

- · The program is basically three parts
- MEM[0...2] increment the elements,
- MEM[0...2] Interinct the centerto,
 MEM[0] and MEM[0] and MEM[2], and
- MEM[9...11] control iteration of the loop body,

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[16]$ 220016 300013 $A \leftarrow A + MEM[13]$ 2 210016 $MEM[16] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state execute 6 220002 $A \leftarrow MEM[2]$ = PC 7 300013 $A \leftarrow A + MEM[13]$ 8 210002 $MEM[2] \leftarrow A$ 220016 9 $= A \leftarrow MEM[16]$ 220000 $A \leftarrow MEM[0]$ 10 310014 $A \leftarrow A - MEM[14]$ = 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 nop 17 nop



University of BRISTOL

git # c144985a @ 2024-10-1

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

Example	(increment e	lements in	the sequ	ence $X =$	$\langle 0, 1, 2 \rangle$
					M

		CPU
state	=	fetch
PC	=	1
IR	=	300013
	=	
Α	=	1

	MEM			
Ī	Address	Value	Semantics	
Ī	0	220016	$A \leftarrow MEM[16]$	
	1	300013	$A \leftarrow A + MEM[13]$	
l	2	210016	$MEM[16] \leftarrow A$	
l	3	220000	$A \leftarrow MEM[0]$	
ı	4	300013	$A \leftarrow A + MEM[13]$	
l	5	210000	$MEM[0] \leftarrow A$	
	6	220002	$A \leftarrow MEM[2]$	
ı	7	300013	$A \leftarrow A + MEM[13]$	
l	8	210002	$MEM[2] \leftarrow A$	
	9	220000	$A \leftarrow MEM[0]$	
	10	310014	$A \leftarrow A - MEM[14]$	
	11	420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$	
ı	12	100000	halt	
l	13	1	nop	
	14	220018	$A \leftarrow MEM[18]$	
	15	1	nop	
	16	1	nop	
	17	2	nop	





Notes:

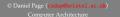
- · The program is basically three parts
- MEM[0...2] increment the elements,
- 2. MEM[3...8] self-modify the instructions in MEM[0] and MEM[2], and
- MEM[9...11] control iteration of the loop body,

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

Notes:

- · The program is basically three parts
 - 1. MEM[0...2] increment the elements,
 - MEM[0...2] Interinct the centerto,
 MEM[0] and MEM[0] and MEM[2], and
- MEM[9...11] control iteration of the loop body,

Example (increment elements in the sequence X = (0, 1, 2)) Address Value Semantics $A \leftarrow MEM[16]$ 220016 300013 $A \leftarrow A + MEM[13]$ MEM[16] ← A 2 210016 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = decode 6 220002 $A \leftarrow MEM[2]$ PC 2 7 300013 $A \leftarrow A + MEM[13]$ 8 210002 $MEM[2] \leftarrow A$ 300013 $A \leftarrow A + MEM[13]$ 9 220000 $A \leftarrow MEM[0]$ 10 310014 $A \leftarrow A - MEM[14]$ = 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 nop 17 nop



University of BRISTOL

git # c144985a @ 2024-10-1

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

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

	CPU
=	execute
=	2
=	300013
=	$A \leftarrow A + MEM[13]$
=	2

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

NAENA





Notes:

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

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

Notes:

- · The program is basically three parts
 - 1. MEM[0...2] increment the elements,
 - MEM[0...2] Interinct the centerto,
 MEM[0] and MEM[0] and MEM[2], and
- MEM[9...11] control iteration of the loop body,

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$) Value Address Semantics $A \leftarrow MEM[16]$ 220016 300013 $A \leftarrow A + MEM[13]$ 2 210016 $MEM[16] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = fetch 6 220002 $A \leftarrow MEM[2]$ PC 2 7 $A \leftarrow A + MEM[13]$ 300013 8 210002 $MEM[2] \leftarrow A$ = 210016 9 220000 $A \leftarrow MEM[0]$ 10 $A \leftarrow A - MEM[14]$ = 2 310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 nop 14 220018 $A \leftarrow MEM[18]$ 15 nop 16 nop 17 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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

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

 $\begin{array}{cccc} & & & & & \\ \text{State} & = & \text{decode} \\ \text{PC} & = & 3 \\ \text{IR} & = & 210016 \\ & = & \text{MEM}[16] \leftarrow \text{A} \\ \text{A} & = & 2 \\ \end{array}$

	MEM					
4	Address Value		Semantics			
	0	220016	$A \leftarrow MEM[16]$			
	1	300013	$A \leftarrow A + MEM[13]$			
	2	210016	$MEM[16] \leftarrow A$			
	3	220000	$A \leftarrow MEM[0]$			
	4	300013	$A \leftarrow A + MEM[13]$			
	5	210000	$MEM[0] \leftarrow A$			
	6	220002	$A \leftarrow MEM[2]$			
	7	300013	$A \leftarrow A + MEM[13]$			
	8	210002	$MEM[2] \leftarrow A$			
	9	220000	$A \leftarrow MEM[0]$			
	10	310014	$A \leftarrow A - MEM[14]$			
	11	420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$			
	12	100000	halt			
	13	1	nop			
	14	220018	$A \leftarrow MEM[18]$			
	15	1	nop			
	16	1	nop			
	17	2	nop			
		, and the second				





Notes:

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

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

Notes:

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

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$) Value Address Semantics $A \leftarrow MEM[16]$ 220016 300013 $A \leftarrow A + MEM[13]$ 2 210016 $MEM[16] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state execute 6 220002 $A \leftarrow MEM[2]$ = PC 7 $A \leftarrow A + MEM[13]$ 3 300013 8 210002 $MEM[2] \leftarrow A$ 210016 9 = $MEM[16] \leftarrow A$ 220000 $A \leftarrow MEM[0]$ 2 10 $A \leftarrow A - MEM[14]$ = 310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 nop 14 220018 $A \leftarrow MEM[18]$ 15 nop 16 2 nop 17 2 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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	$A \leftarrow MEM[16]$				
1	300013	$A \leftarrow A + MEM[13]$				
2	210016	$MEM[16] \leftarrow A$				
3	220000	$A \leftarrow MEM[0]$				
4	300013	$A \leftarrow A + MEM[13]$				
5	210000	$MEM[0] \leftarrow A$				
6	220002	$A \leftarrow MEM[2]$				
7	300013	$A \leftarrow A + MEM[13]$				
8	210002	$MEM[2] \leftarrow A$				
9	220000	$A \leftarrow MEM[0]$				
10	310014	$A \leftarrow A - MEM[14]$				
11	420000	$PC \leftarrow 0 \text{ iff. A} \neq 0$				
12	100000	halt				
13	1	nop				
14	220018	$A \leftarrow MEM[18]$				
15	1	nop				
16	2	nop				
17	2	nop				
	•	_				





Notes:

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

where $\mathsf{MEM}[13\dots14]$ are constants, and X is held in $\mathsf{MEM}[15\dots17]$.

Notes:

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

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$) Value Address Semantics $A \leftarrow MEM[16]$ 220016 300013 $A \leftarrow A + MEM[13]$ 2 210016 $MEM[16] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = decode 6 220002 $A \leftarrow MEM[2]$ PC 7 $A \leftarrow A + MEM[13]$ 300013 8 210002 $MEM[2] \leftarrow A$ 220000 9 $= A \leftarrow MEM[0]$ 220000 $A \leftarrow MEM[0]$ 2 10 $A \leftarrow A - MEM[14]$ = 310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 nop 14 220018 $A \leftarrow MEM[18]$ 15 nop 16 2 nop 17 2 nop



University of BRISTOL

git # c144985a @ 2024-10-1

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	$A \leftarrow MEM[16]$	
1	300013	$A \leftarrow A + MEM[13]$	
2	210016	MEM[16] ← A	
3	220000	$A \leftarrow MEM[0]$	
4	300013	$A \leftarrow A + MEM[13]$	
5	210000	MEM[0] ← A	
6	220002	$A \leftarrow MEM[2]$	
7	300013	$A \leftarrow A + MEM[13]$	
8	210002	MEM[2] ← A	
9	220000	$A \leftarrow MEM[0]$	
10	310014	$A \leftarrow A - MEM[14]$	
11	420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$	
12	100000	halt	
13	1	nop	
14	220018	$A \leftarrow MEM[18]$	
15	1	nop	
16	2	nop	
17	2	nop	
		_	





Notes:

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

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

Notes:

The program is basically three parts

MEM[0...2] increment the elements,

MEM[3...8] self-modify the instructions in MEM[0] and MEM[2], and

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)

	Address	М		
	Addrose		EM	
	Addiess	Value	Semantics	
	0	220016	$A \leftarrow MEM[16]$	
	1	300013	$A \leftarrow A + MEM[13]$	
	2	210016	$MEM[16] \leftarrow A$	
	3	220000	$A \leftarrow MEM[0]$	
	4	300013	$A \leftarrow A + MEM[13]$	
CPU	5	210000	$MEM[0] \leftarrow A$	
state = fetch	6	220002	$A \leftarrow MEM[2]$	
PC = 4	7	300013	$A \leftarrow A + MEM[13]$	
IR = 300013	8	210002	MEM[2] ← A	
=	9	220000	$A \leftarrow MEM[0]$	
A = 220016	10	310014	$A \leftarrow A - MEM[14]$	
	11	420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$	
	12	100000	halt	
	13	1	nop	
	14	220018	$A \leftarrow MEM[18]$	
	15	1	nop	
	16	2	nop	
	17	2	nop	



University of BRISTOL

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

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

		CPU	
state	=		

		CPU
state	=	decode
PC	=	5
IR	=	300013
	=	$A \leftarrow A + MEM[13]$
Α	=	220016

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





Notes:

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

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

- 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 $\mathsf{MEM}[13 \dots 14]$ are constants, and X is held in $\mathsf{MEM}[15 \dots 17]$.

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[16]$ 220016 300013 $A \leftarrow A + MEM[13]$ 2 210016 $MEM[16] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state execute 6 220002 $A \leftarrow MEM[2]$ = PC 7 $A \leftarrow A + MEM[13]$ 300013 8 210002 $MEM[2] \leftarrow A$ 300013 $A \leftarrow A + MEM[13]$ 9 220000 $A \leftarrow MEM[0]$ 220017 10 310014 $A \leftarrow A - MEM[14]$ 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 2 17 nop



University of BRISTOL

git # c144985a @ 2024-10-1

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

Example (increment elements in the sec	querice A	- (0,
_		
	Addrose	Valu

		CPU
state	=	fetch
PC	=	5
IR	=	210000
	=	
Α	=	220017

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





Notes:

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

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

Notes:

- · The program is basically three parts
 - 1. MEM[0...2] increment the elements,
 - MEM[0...2] Interinct the centerto,
 MEM[0] and MEM[0] and MEM[2], and
- MEM[9...11] control iteration of the loop body,

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[16]$ 220016 300013 $A \leftarrow A + MEM[13]$ 2 210016 $MEM[16] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = decode 6 220002 $A \leftarrow MEM[2]$ PC 7 300013 $A \leftarrow A + MEM[13]$ 8 210002 $MEM[2] \leftarrow A$ 210000 9 $MEM[0] \leftarrow A$ 220000 $A \leftarrow MEM[0]$ 220017 10 310014 $A \leftarrow A - MEM[14]$ 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 17 2 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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

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

 CPU

 state
 = execute

 PC
 = 6

 IR
 = 210000

 = MEM[0] ← A

 A
 = 220017

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





Notes:

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

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

Notes

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

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[17]$ 220017 300013 $A \leftarrow A + MEM[13]$ 2 210016 $MEM[16] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = fetch 6 220002 $A \leftarrow MEM[2]$ PC 7 300013 $A \leftarrow A + MEM[13]$ 8 210002 $MEM[2] \leftarrow A$ = 220002 9 220000 $A \leftarrow MEM[0]$ 220017 10 310014 $A \leftarrow A - MEM[14]$ = 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 17 2 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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

	ME	-M
Address	Value	Semantics
0	220017	$A \leftarrow MEM[17]$
1	300013	$A \leftarrow A + MEM[13]$
2	210016	MEM[16] ← A
3	220000	$A \leftarrow MEM[0]$
4	300013	$A \leftarrow A + MEM[13]$
5	210000	$MEM[0] \leftarrow A$
6	220002	$A \leftarrow MEM[2]$
7	300013	$A \leftarrow A + MEM[13]$
8	210002	MEM[2] ← A
9	220000	$A \leftarrow MEM[0]$
10	310014	$A \leftarrow A - MEM[14]$
11	420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$
12	100000	halt
13	1	nop
14	220018	$A \leftarrow MEM[18]$
15	1	nop
16	2	nop
17	2	nop





Notes:

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

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

Notes:

- · The program is basically three parts
 - 1. MEM[0...2] increment the elements,
 - MEM[0...2] Interinct the centerto,
 MEM[0] and MEM[0] and MEM[2], and
- MEM[9...11] control iteration of the loop body,

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[17]$ 220017 300013 $A \leftarrow A + MEM[13]$ 2 210016 $MEM[16] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state execute 6 220002 $A \leftarrow MEM[2]$ = PC 7 $A \leftarrow A + MEM[13]$ 300013 8 $MEM[2] \leftarrow A$ 220002 210002 9 $A \leftarrow MEM[2]$ 220000 $A \leftarrow MEM[0]$ 210016 10 $A \leftarrow A - MEM[14]$ = 310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 17 2 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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

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

		CPU	
state	=	CPU fetch	
PC	= =	fetch 7	
		fetch	
PC	=	fetch 7	

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





Notes:

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

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

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 $X = \langle 0, 1, 2 \rangle$) Value Address Semantics $A \leftarrow MEM[17]$ 220017 300013 $A \leftarrow A + MEM[13]$ 2 210016 $MEM[16] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = decode 6 220002 $A \leftarrow MEM[2]$ PC 7 $A \leftarrow A + MEM[13]$ 300013 8 210002 $MEM[2] \leftarrow A$ 300013 9 $A \leftarrow A + MEM[13]$ 220000 $A \leftarrow MEM[0]$ 210016 10 $A \leftarrow A - MEM[14]$ 310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 17 2 nop



University of BRISTOL

git # c144985a @ 2024-10-1

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	

= 210017

 $A \leftarrow A + MEM[13]$

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





Notes:

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

```
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 = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[17]$ 220017 300013 $A \leftarrow A + MEM[13]$ 2 210016 $MEM[16] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = fetch 6 220002 $A \leftarrow MEM[2]$ PC 7 300013 $A \leftarrow A + MEM[13]$ 8 210002 $MEM[2] \leftarrow A$ = 210002 9 220000 $A \leftarrow MEM[0]$ 210017 10 310014 $A \leftarrow A - MEM[14]$ = 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 nop 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 17 2 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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

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

 CPU

 state
 =
 decode

 PC
 =
 9

 IR
 =
 210002

 =
 MEM[2] ← A

 A
 =
 210017

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	MEM				
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	Address	Value	Semantics		
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	0	220017	$A \leftarrow MEM[17]$		
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	1	300013	$A \leftarrow A + MEM[13]$		
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	2	210016	$MEM[16] \leftarrow A$		
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	3	220000	$A \leftarrow MEM[0]$		
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	4	300013	$A \leftarrow A + MEM[13]$		
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	5	210000	$MEM[0] \leftarrow A$		
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	6	220002	$A \leftarrow MEM[2]$		
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	7	300013	$A \leftarrow A + MEM[13]$		
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	8	210002	$MEM[2] \leftarrow A$		
11 420000 PC \leftarrow 0 iff. A \neq 0 halt	9	220000	$A \leftarrow MEM[0]$		
12 100000 halt	10	310014	$A \leftarrow A - MEM[14]$		
	11	420000	$PC \leftarrow 0 \text{ iff. A} \neq 0$		
13 1 nop	12	100000	halt		
	13	1	nop		
14 220018 $A \leftarrow MEM[18]$	14	220018	$A \leftarrow MEM[18]$		
15 1 nop	15	1	nop		
16 2 nop	16	2	nop		
17 2 nop	17	2	nop		





Notes:

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

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

Notes:

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

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[17]$ 220017 300013 $A \leftarrow A + MEM[13]$ 2 210017 $MEM[17] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state execute 6 220002 $A \leftarrow MEM[2]$ = PC 7 $A \leftarrow A + MEM[13]$ 300013 8 210002 $MEM[2] \leftarrow A$ 210002 9 = $MEM[2] \leftarrow A$ 220000 $A \leftarrow MEM[0]$ 210017 10 $A \leftarrow A - MEM[14]$ 310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 nop 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 17 2 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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

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

	CDLI	1

		CPU
state	=	fetch
PC	=	9
IR	=	220000
	=	
Α	=	210017

$ \begin{array}{c ccccc} Address & Value & Semantics \\ \hline 0 & 220017 & A \leftarrow MEM[17] \\ 1 & 300013 & A \leftarrow A + MEM[13] \\ 2 & 210017 & MEM[17] \leftarrow A \\ 3 & 220000 & A \leftarrow MEM[0] \\ 4 & 300013 & A \leftarrow A + MEM[13] \\ 5 & 210000 & MEM[0] \leftarrow A \\ \hline \end{array} $	
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	
5 210000 MEM[0] ← A	
[]	
$6 \qquad 220002 \qquad A \leftarrow MEM[2]$	
7 300013 $A \leftarrow A + MEM[13]$	
8 210002 $MEM[2] \leftarrow A$	
9 220000 $A \leftarrow MEM[0]$	
10 310014 $A \leftarrow A - MEM[14]$	
11 420000 PC \leftarrow 0 iff. A \neq 0	
12 100000 halt	
13 1 nop	
14 220018 $A \leftarrow MEM[18]$	
15 1 nop	
16 2 nop	
17 2 nop	





Notes:

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

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

Notes:

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

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$) Value Address Semantics $A \leftarrow MEM[17]$ 220017 300013 $A \leftarrow A + MEM[13]$ 2 210017 $MEM[17] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = decode 6 220002 $A \leftarrow MEM[2]$ PC 10 7 $A \leftarrow A + MEM[13]$ 300013 8 $MEM[2] \leftarrow A$ 220000 210002 9 $A \leftarrow MEM[0]$ 220000 $A \leftarrow MEM[0]$ 210017 10 $A \leftarrow A - MEM[14]$ = 310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 nop 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 17 2 nop



University of BRISTOL

git # c144985a @ 2024-10-14

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	$A \leftarrow MEM[17]$				
1	300013	$A \leftarrow A + MEM[13]$				
2	210017	$MEM[17] \leftarrow A$				
3	220000	$A \leftarrow MEM[0]$				
4	300013	$A \leftarrow A + MEM[13]$				
5	210000	$MEM[0] \leftarrow A$				
6	220002	$A \leftarrow MEM[2]$				
7	300013	$A \leftarrow A + MEM[13]$				
8	210002	$MEM[2] \leftarrow A$				
9	220000	$A \leftarrow MEM[0]$				
10	310014	$A \leftarrow A - MEM[14]$				
11	420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$				
12	100000	halt				
13	1	nop				
14	220018	$A \leftarrow MEM[18]$				
15	1	nop				
16	2	nop				
17	2	nop				





Notes:

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

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

Notes:

- · The program is basically three parts
 - MEM[0...2] increment the elements,
 - MEM[0...2] Interinct the centerto,
 MEM[0] and MEM[0] and MEM[2], and
- MEM[9...11] control iteration of the loop body,

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[17]$ 220017 300013 $A \leftarrow A + MEM[13]$ 2 210017 $MEM[17] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = fetch 6 220002 $A \leftarrow MEM[2]$ PC 10 7 $A \leftarrow A + MEM[13]$ 300013 8 $MEM[2] \leftarrow A$ = 310014 210002 9 220000 $A \leftarrow MEM[0]$ 220017 10 $A \leftarrow A - MEM[14]$ = 310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 17 2 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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
	=	$A \leftarrow A - MEM[14]$

= 220017

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





Notes:

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

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

Notes:

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

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[17]$ 220017 300013 $A \leftarrow A + MEM[13]$ 2 210017 $MEM[17] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state execute 6 220002 $A \leftarrow MEM[2]$ = PC 7 300013 $A \leftarrow A + MEM[13]$ 11 8 210002 $MEM[2] \leftarrow A$ 310014 $A \leftarrow A - MEM[14]$ 9 220000 $A \leftarrow MEM[0]$ 10 310014 $A \leftarrow A - MEM[14]$ = -111 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 17 2 nop



University of BRISTOL

git # c144985a @ 2024-10-1

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

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

		CPU
state	=	fetch
PC	=	11
IR	=	420000
	=	
Α	=	-1
•		

MEM					
Address	Value	Semantics			
0	220017	$A \leftarrow MEM[17]$			
1	300013	$A \leftarrow A + MEM[13]$			
2	210017	$MEM[17] \leftarrow A$			
3	220000	$A \leftarrow MEM[0]$			
4	300013	$A \leftarrow A + MEM[13]$			
5	210000	$MEM[0] \leftarrow A$			
6	220002	$A \leftarrow MEM[2]$			
7	300013	$A \leftarrow A + MEM[13]$			
8	210002	$MEM[2] \leftarrow A$			
9	220000	$A \leftarrow MEM[0]$			
10	310014	$A \leftarrow A - MEM[14]$			
11	420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$			
12	100000	halt			
13	1	nop			
14	220018	$A \leftarrow MEM[18]$			
15	1	nop			
16	2	nop			
17	2	nop			
		•			





Notes:

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

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

Notes:

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

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

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$)						
					N	1EM
				Address	Value	Semantics
				0	220017	$A \leftarrow MEM[17]$
				1	300013	$A \leftarrow A + MEM[13]$
				2	210017	$MEM[17] \leftarrow A$
				3	220000	$A \leftarrow MEM[0]$
				4	300013	$A \leftarrow A + MEM[13]$
		CPU		5	210000	$MEM[0] \leftarrow A$
state	=	decode		6	220002	$A \leftarrow MEM[2]$
PC	=	12		7	300013	$A \leftarrow A + MEM[13]$
IR	=	420000		8	210002	$MEM[2] \leftarrow A$
	=	$PC \leftarrow 0 \text{ iff. } A \neq 0$		9	220000	$A \leftarrow MEM[0]$
Α	=	-1		10	310014	$A \leftarrow A - MEM[14]$
				11	420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$
				12	100000	halt
				13	1	nop
				14	220018	$A \leftarrow MEM[18]$
				15	1	nop
				16	2	nop
				17	2	nop



University of BRISTOL

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

|--|

		CPU
state	=	execute
PC	=	0
IR	=	420000
	=	$PC \leftarrow 0 \text{ iff. } A \neq 0$
Α	=	-1

MEM Address Value Semantics					
Value	Semantics				
220017	$A \leftarrow MEM[17]$				
300013	$A \leftarrow A + MEM[13]$				
210017	$MEM[17] \leftarrow A$				
220000	$A \leftarrow MEM[0]$				
300013	$A \leftarrow A + MEM[13]$				
210000	$MEM[0] \leftarrow A$				
220002	$A \leftarrow MEM[2]$				
300013	$A \leftarrow A + MEM[13]$				
210002	$MEM[2] \leftarrow A$				
220000	$A \leftarrow MEM[0]$				
310014	$A \leftarrow A - MEM[14]$				
420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$				
100000	halt				
1	nop				
220018	$A \leftarrow MEM[18]$				
1	nop				
2	nop				
2	nop				
	Value 220017 300013 210017 220000 300013 210000 220002 300013 210002 220000 310014 420000 100000 1 220018 1 2				





Notes:

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

where $\mathsf{MEM}[13 \dots 14]$ are constants, and X is held in $\mathsf{MEM}[15 \dots 17]$.

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

where $\mathsf{MEM}[13 \dots 14]$ are constants, and X is held in $\mathsf{MEM}[15 \dots 17]$.

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$) Value Address Semantics $A \leftarrow MEM[17]$ 220017 300013 $A \leftarrow A + MEM[13]$ 2 210017 $MEM[17] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = fetch 6 220002 $A \leftarrow MEM[2]$ PC 7 $A \leftarrow A + MEM[13]$ 300013 8 210002 $MEM[2] \leftarrow A$ = 220017 9 220000 $A \leftarrow MEM[0]$ 10 $A \leftarrow A - MEM[14]$ = -1310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 nop 14 220018 $A \leftarrow MEM[18]$ 15 nop 16 2 nop 17 2 nop





git # c144985a @ 2024-10-1-

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

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

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





Notes:

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

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

Notes:

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

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[17]$ 220017 300013 $A \leftarrow A + MEM[13]$ 2 210017 $MEM[17] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state execute 6 220002 $A \leftarrow MEM[2]$ = PC 7 300013 $A \leftarrow A + MEM[13]$ 8 210002 $MEM[2] \leftarrow A$ 220017 9 $= A \leftarrow MEM[17]$ 220000 $A \leftarrow MEM[0]$ 2 10 310014 $A \leftarrow A - MEM[14]$ = 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 nop 16 2 nop 17 2 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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

Exampl	le (increment e	lements in t	he sequence	$X = \langle 0, 1, 2 \rangle)$

CPU

state = fetch
PC = 1
IR = 300013
= A = 2

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





Notes:

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

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

Notes:

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

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[17]$ 220017 300013 $A \leftarrow A + MEM[13]$ 2 210017 $MEM[17] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = decode 6 220002 $A \leftarrow MEM[2]$ PC 2 7 $A \leftarrow A + MEM[13]$ 300013 8 210002 $MEM[2] \leftarrow A$ 300013 $= A \leftarrow A + MEM[13]$ 9 220000 $A \leftarrow MEM[0]$ 2 10 310014 $A \leftarrow A - MEM[14]$ = 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 17 2 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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

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

	CPU
=	execute
=	2
=	300013
=	$A \leftarrow A + MEM[13]$
=	3

IVILIVI			
Address Value		Semantics	
0	220017	$A \leftarrow MEM[17]$	
1	300013	$A \leftarrow A + MEM[13]$	
2	210017	$MEM[17] \leftarrow A$	
3	220000	$A \leftarrow MEM[0]$	
4	300013	$A \leftarrow A + MEM[13]$	
5	210000	$MEM[0] \leftarrow A$	
6	220002	$A \leftarrow MEM[2]$	
7	300013	$A \leftarrow A + MEM[13]$	
8	210002	$MEM[2] \leftarrow A$	
9	220000	$A \leftarrow MEM[0]$	
10	310014	$A \leftarrow A - MEM[14]$	
11	420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$	
12	100000	halt	
13	1	nop	
14	220018	$A \leftarrow MEM[18]$	
15	1	nop	
16	2	nop	
17	2	nop	

MFM





Notes:

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

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

• The

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

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$) Value Address Semantics $A \leftarrow MEM[17]$ 220017 300013 $A \leftarrow A + MEM[13]$ 2 210017 $MEM[17] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = fetch 6 220002 $A \leftarrow MEM[2]$ PC 2 7 $A \leftarrow A + MEM[13]$ 300013 8 210002 $MEM[2] \leftarrow A$ = 210017 9 220000 $A \leftarrow MEM[0]$ 10 $A \leftarrow A - MEM[14]$ = 3 310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 nop 14 220018 $A \leftarrow MEM[18]$ 15 nop 16 2 nop 17 2 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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	$A \leftarrow MEM[17]$		
1	300013	$A \leftarrow A + MEM[13]$		
2	210017	MEM[17] ← A		
3	220000	$A \leftarrow MEM[0]$		
4	300013	$A \leftarrow A + MEM[13]$		
5	210000	$MEM[0] \leftarrow A$		
6	220002	$A \leftarrow MEM[2]$		
7	300013	$A \leftarrow A + MEM[13]$		
8	210002	$MEM[2] \leftarrow A$		
9	220000	$A \leftarrow MEM[0]$		
10	310014	$A \leftarrow A - MEM[14]$		
11	420000	$PC \leftarrow 0 \text{ iff. A} \neq 0$		
12	100000	halt		
13	1	nop		
14	220018	$A \leftarrow MEM[18]$		
15	1	nop		
16	2	nop		
17	2	nop		





Notes:

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

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

Notes:

- · The program is basically three parts
 - MEM[0...2] increment the elements,
 - MEM[0...2] Interinct the centerto,
 MEM[0] and MEM[0] and MEM[2], and
- MEM[9...11] control iteration of the loop body,

Example (increment elements in the sequence X = (0, 1, 2)) Address Value Semantics $A \leftarrow MEM[17]$ 220017 1 300013 $A \leftarrow A + MEM[13]$ MEM[17] ← A 2 210017 3 220000 $A \leftarrow MEM[0]$ 4 $A \leftarrow A + MEM[13]$ 300013 CPU 5 210000 $MEM[0] \leftarrow A$ 6 $A \leftarrow MEM[2]$ state = execute 220002 $A \leftarrow A + MEM[13]$ PC 3 7 300013 8 $MEM[2] \leftarrow A$ 210017 210002 9 $A \leftarrow MEM[0]$ = MEM[17] ← A 220000 = 3 10 310014 $A \leftarrow A - MEM[14]$ 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 nop 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 17 3 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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

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

		CPU
state	=	fetch
PC	=	3
IR	=	220000
	=	
Α	=	3

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





Notes:

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

where $\mathsf{MEM}[13\dots14]$ are constants, and X is held in $\mathsf{MEM}[15\dots17]$.

No	tes:
•	The program is basically three parts
	MEM[0,2] increment the elements, MEM[38] self-modify the instructions in MEM[0] and MEM[2], and MEM[911] control iteration of the loop body,
	where $MEM[13\dots14]$ are constants, and X is held in $MEM[15\dots17]$.

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[17]$ 220017 300013 $A \leftarrow A + MEM[13]$ 2 210017 $MEM[17] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = decode 6 220002 $A \leftarrow MEM[2]$ PC 7 300013 $A \leftarrow A + MEM[13]$ 8 210002 $MEM[2] \leftarrow A$ 220000 9 $= A \leftarrow MEM[0]$ 220000 $A \leftarrow MEM[0]$ 3 10 310014 $A \leftarrow A - MEM[14]$ = 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 nop 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 17 3 nop



University of BRISTOL

git # c144985a @ 2024-10-1

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

Example (increment el	ements in	the sequ	ience X	$=\langle 0,1\rangle$,2))
			_			

 CPU

 state
 =
 execute

 PC
 =
 4

 IR
 =
 220000

 =
 A ← MEM[0]

 A
 =
 220017

MEM			
Address	Value	Semantics	
0	220017	$A \leftarrow MEM[17]$	
1	300013	$A \leftarrow A + MEM[13]$	
2	210017	$MEM[17] \leftarrow A$	
3	220000	$A \leftarrow MEM[0]$	
4	300013	$A \leftarrow A + MEM[13]$	
5	210000	$MEM[0] \leftarrow A$	
6	220002	$A \leftarrow MEM[2]$	
7	300013	$A \leftarrow A + MEM[13]$	
8	210002	$MEM[2] \leftarrow A$	
9	220000	$A \leftarrow MEM[0]$	
10	310014	$A \leftarrow A - MEM[14]$	
11	420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$	
12	100000	halt	
13	1	nop	
14	220018	$A \leftarrow MEM[18]$	
15	1	nop	
16	2	nop	
17	3	nop	
•			





Notes:

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

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

Notes:

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

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[17]$ 220017 300013 $A \leftarrow A + MEM[13]$ 2 210017 $MEM[17] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = fetch 6 220002 $A \leftarrow MEM[2]$ PC 7 $A \leftarrow A + MEM[13]$ 300013 8 210002 $MEM[2] \leftarrow A$ = 300013 9 220000 $A \leftarrow MEM[0]$ 220017 10 $A \leftarrow A - MEM[14]$ 310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 17 3 nop





git # c144985a @ 2024-10-1-

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
	=	$A \leftarrow A + MEM[13]$
Α	=	220017
•		

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





Notes:

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

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

Notes:

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

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[17]$ 220017 300013 $A \leftarrow A + MEM[13]$ 2 210017 $MEM[17] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state execute 6 220002 $A \leftarrow MEM[2]$ = PC 5 7 300013 $A \leftarrow A + MEM[13]$ 8 210002 $MEM[2] \leftarrow A$ 300013 9 $A \leftarrow A + MEM[13]$ 220000 $A \leftarrow MEM[0]$ 220018 10 $A \leftarrow A - MEM[14]$ 310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 nop 16 2 nop 17 3 nop



University of BRISTOL

git # c144985a @ 2024-10-1

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
	=	
Α	=	220018
	PC IR	PC = IR = =

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





Notes:

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

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

Notes:

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

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[17]$ 220017 300013 $A \leftarrow A + MEM[13]$ 2 210017 $MEM[17] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = decode 6 220002 $A \leftarrow MEM[2]$ PC 7 300013 $A \leftarrow A + MEM[13]$ 8 210002 $MEM[2] \leftarrow A$ 210000 9 $MEM[0] \leftarrow A$ 220000 $A \leftarrow MEM[0]$ 220018 10 310014 $A \leftarrow A - MEM[14]$ 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 nop 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 17 3 nop



University of BRISTOL

git # c144985a @ 2024-10-1

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

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

 CPU

 state
 =
 execute

 PC
 =
 6

 IR
 =
 210000

 =
 MEM[0] ← A

 A
 =
 220018

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





Notes:

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

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

Notes:

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

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[18]$ 220018 300013 $A \leftarrow A + MEM[13]$ 2 210017 $MEM[17] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = fetch 6 220002 $A \leftarrow MEM[2]$ PC 7 $A \leftarrow A + MEM[13]$ 300013 8 210002 $MEM[2] \leftarrow A$ = 220002 9 220000 $A \leftarrow MEM[0]$ 220018 10 310014 $A \leftarrow A - MEM[14]$ = 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 nop 14 220018 $A \leftarrow MEM[18]$ 15 nop 16 2 nop 17 3 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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

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

 CPU

 state
 =
 decode

 PC
 =
 7

 IR
 =
 220002

 =
 A ← MEM[2]

 A
 =
 220018

	MEM			
Address	Value	Semantics		
0	220018	$A \leftarrow MEM[18]$		
1	300013	$A \leftarrow A + MEM[13]$		
2	210017	$MEM[17] \leftarrow A$		
3	220000	$A \leftarrow MEM[0]$		
4	300013	$A \leftarrow A + MEM[13]$		
5	210000	$MEM[0] \leftarrow A$		
6	220002	$A \leftarrow MEM[2]$		
7	300013	$A \leftarrow A + MEM[13]$		
8	210002	$MEM[2] \leftarrow A$		
9	220000	$A \leftarrow MEM[0]$		
10	310014	$A \leftarrow A - MEM[14]$		
11	420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$		
12	100000	halt		
13	1	nop		
14	220018	$A \leftarrow MEM[18]$		
15	1	nop		
16	2	nop		
17	3	nop		





Notes:

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

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

Notes:

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

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[18]$ 220018 300013 $A \leftarrow A + MEM[13]$ 2 210017 $MEM[17] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state execute 6 220002 $A \leftarrow MEM[2]$ = PC 7 300013 $A \leftarrow A + MEM[13]$ 8 210002 $MEM[2] \leftarrow A$ 220002 9 $A \leftarrow MEM[2]$ 220000 $A \leftarrow MEM[0]$ 210017 10 310014 $A \leftarrow A - MEM[14]$ = 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 nop 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 17 3 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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

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

CPU

state = fetch
PC = 7
IR = 300013
= =
A = 210017

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





Notes:

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

where $\mathsf{MEM}[13\dots14]$ are constants, and X is held in $\mathsf{MEM}[15\dots17]$.

es: The program is

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

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$) Value Address Semantics $A \leftarrow MEM[18]$ 220018 300013 $A \leftarrow A + MEM[13]$ 2 210017 $MEM[17] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = decode 6 220002 $A \leftarrow MEM[2]$ PC 7 $A \leftarrow A + MEM[13]$ 300013 8 $MEM[2] \leftarrow A$ 300013 210002 $A \leftarrow A + MEM[13]$ 9 220000 $A \leftarrow MEM[0]$ 210017 10 $A \leftarrow A - MEM[14]$ 310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 17 3 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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	

= 210018

 $A \leftarrow A + MEM[13]$

MEM				
Address	Value	Semantics		
0	220018	$A \leftarrow MEM[18]$		
1	300013	$A \leftarrow A + MEM[13]$		
2	210017	$MEM[17] \leftarrow A$		
3	220000	$A \leftarrow MEM[0]$		
4	300013	$A \leftarrow A + MEM[13]$		
5	210000	$MEM[0] \leftarrow A$		
6	220002	$A \leftarrow MEM[2]$		
7	300013	$A \leftarrow A + MEM[13]$		
8	210002	$MEM[2] \leftarrow A$		
9	220000	$A \leftarrow MEM[0]$		
10	310014	$A \leftarrow A - MEM[14]$		
11	420000	$PC \leftarrow 0 \text{ iff. A} \neq 0$		
12	100000	halt		
13	1	nop		
14	220018	$A \leftarrow MEM[18]$		
15	1	nop		
16	2	nop		
17	3	nop		





Notes:

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

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

Notes:

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

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[18]$ 220018 300013 $A \leftarrow A + MEM[13]$ 2 210017 $MEM[17] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = fetch 6 220002 $A \leftarrow MEM[2]$ PC 7 300013 $A \leftarrow A + MEM[13]$ 8 210002 $MEM[2] \leftarrow A$ = 210002 9 220000 $A \leftarrow MEM[0]$ 210018 10 310014 $A \leftarrow A - MEM[14]$ = 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 17 3 nop



University of BRISTOL

git # c144985a @ 2024-10-1

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

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

		CPU
state	=	decode
PC	=	9
IR	=	210002
	=	$MEM[2] \leftarrow A$
Α	=	210018

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





Notes:

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

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

Notes:

- · The program is basically three parts
- MEM[0...2] increment the elements,
- MEM[0...2] Interinct the centerto,
 MEM[0] and MEM[0] and MEM[2], and
- MEM[9...11] control iteration of the loop body,

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[18]$ 220018 300013 $A \leftarrow A + MEM[13]$ MEM[18] ← A 2 210018 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state execute 6 220002 $A \leftarrow MEM[2]$ = PC 7 $A \leftarrow A + MEM[13]$ 300013 8 210002 $MEM[2] \leftarrow A$ 210002 9 $MEM[2] \leftarrow A$ 220000 $A \leftarrow MEM[0]$ 210018 10 $A \leftarrow A - MEM[14]$ 310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 nop 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 17 3 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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

Example (increment elements in the sequence X = (0, 1, 2))

		-

		CPU
state	=	fetch
PC	=	9
IR	=	220000
	=	
Α	=	210018

MEM			
Address	Value	Semantics	
0	220018	$A \leftarrow MEM[18]$	
1	300013	$A \leftarrow A + MEM[13]$	
2	210018	$MEM[18] \leftarrow A$	
3	220000	$A \leftarrow MEM[0]$	
4	300013	$A \leftarrow A + MEM[13]$	
5	210000	$MEM[0] \leftarrow A$	
6	220002	$A \leftarrow MEM[2]$	
7	300013	$A \leftarrow A + MEM[13]$	
8	210002	MEM[2] ← A	
9	220000	$A \leftarrow MEM[0]$	
10	310014	$A \leftarrow A - MEM[14]$	
11	420000	$PC \leftarrow 0 \text{ iff. A} \neq 0$	
12	100000	halt	
13	1	nop	
14	220018	$A \leftarrow MEM[18]$	
15	1	nop	
16	2	nop	
17	3	nop	





Notes:

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

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

Notes:

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

Example (increment elements in the sequence $X = \langle 0, 1, 2 \rangle$) Value Address Semantics $A \leftarrow MEM[18]$ 220018 300013 $A \leftarrow A + MEM[13]$ 2 210018 $MEM[18] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = decode 6 220002 $A \leftarrow MEM[2]$ PC 10 7 $A \leftarrow A + MEM[13]$ 300013 8 $MEM[2] \leftarrow A$ 220000 210002 9 $A \leftarrow MEM[0]$ 220000 $A \leftarrow MEM[0]$ 210018 10 $A \leftarrow A - MEM[14]$ = 310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 nop 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 17 3 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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

Address	Value	Semantics
0	220018	$A \leftarrow MEM[18]$
1	300013	$A \leftarrow A + MEM[13]$
2	210018	$MEM[18] \leftarrow A$
3	220000	$A \leftarrow MEM[0]$
4	300013	$A \leftarrow A + MEM[13]$
5	210000	$MEM[0] \leftarrow A$
6	220002	$A \leftarrow MEM[2]$
7	300013	$A \leftarrow A + MEM[13]$
8	210002	MEM[2] ← A
9	220000	$A \leftarrow MEM[0]$
10	310014	$A \leftarrow A - MEM[14]$
11	420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$
12	100000	halt
13	1	nop
14	220018	$A \leftarrow MEM[18]$
15	1	nop
16	2	nop
17	3	nop
		-

MEM





Notes:

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

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

Notes:

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

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[18]$ 220018 300013 $A \leftarrow A + MEM[13]$ 2 210018 $MEM[18] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = fetch 6 220002 $A \leftarrow MEM[2]$ PC 10 7 $A \leftarrow A + MEM[13]$ 300013 8 210002 $MEM[2] \leftarrow A$ = 310014 9 220000 $A \leftarrow MEM[0]$ 220018 10 $A \leftarrow A - MEM[14]$ = 310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 17 3 nop



University of BRISTOL

git # c144985a @ 2024-10-

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

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

Г			
state	=	CPU decode	

		CPU
state	=	decode
PC	=	11
IR	=	310014
	=	$A \leftarrow A - MEM[14]$
Α	=	220018

MEM			
Address	Value	Semantics	
0	220018	$A \leftarrow MEM[18]$	
1	300013	$A \leftarrow A + MEM[13]$	
2	210018	$MEM[18] \leftarrow A$	
3	220000	$A \leftarrow MEM[0]$	
4	300013	$A \leftarrow A + MEM[13]$	
5	210000	$MEM[0] \leftarrow A$	
6	220002	$A \leftarrow MEM[2]$	
7	300013	$A \leftarrow A + MEM[13]$	
8	210002	MEM[2] ← A	
9	220000	$A \leftarrow MEM[0]$	
10	310014	$A \leftarrow A - MEM[14]$	
11	420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$	
12	100000	halt	
13	1	nop	
14	220018	$A \leftarrow MEM[18]$	
15	1	nop	
16	2	nop	
17	3	nop	





Notes:

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

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

Notes:

- · The program is basically three parts
- MEM[0...2] increment the elements,
- MEM[0...2] Interinct the centerto,
 MEM[0] and MEM[0] and MEM[2], and
- MEM[9...11] control iteration of the loop body,

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

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[18]$ 220018 300013 $A \leftarrow A + MEM[13]$ MEM[18] ← A 2 210018 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state execute 6 220002 $A \leftarrow MEM[2]$ = PC 7 300013 $A \leftarrow A + MEM[13]$ 11 8 210002 $MEM[2] \leftarrow A$ 310014 9 $= A \leftarrow A - MEM[14]$ 220000 $A \leftarrow MEM[0]$ 0 10 310014 $A \leftarrow A - MEM[14]$ = 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 17 3 nop



University of BRISTOL

git # c144985a @ 2024-10-1

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

Example (i	increment	elements in	the sequ	ience X	$=\langle 0,1\rangle$	$,2\rangle$
						N 4 F

CPU
state = fetch
PC = 11
IR = 420000
= A = 0

MEM			
Address	Value	Semantics	
0	220018	$A \leftarrow MEM[18]$	
1	300013	$A \leftarrow A + MEM[13]$	
2	210018	MEM[18] ← A	
3	220000	$A \leftarrow MEM[0]$	
4	300013	$A \leftarrow A + MEM[13]$	
5	210000	$MEM[0] \leftarrow A$	
6	220002	$A \leftarrow MEM[2]$	
7	300013	$A \leftarrow A + MEM[13]$	
8	210002	MEM[2] ← A	
9	220000	$A \leftarrow MEM[0]$	
10	310014	$A \leftarrow A - MEM[14]$	
11	420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$	
12	100000	halt	
13	1	nop	
14	220018	$A \leftarrow MEM[18]$	
15	1	nop	
16	2	nop	
17	3	nop	





Notes:

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

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

The program is basically three parts
 MEM[0...2] increment the elements,
 MEM[3...8] self-modify the instructions in MEM[0] and MEM[2], and

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 = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[18]$ 220018 300013 $A \leftarrow A + MEM[13]$ MEM[18] ← A 2 210018 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = decode 6 220002 $A \leftarrow MEM[2]$ PC 12 7 300013 $A \leftarrow A + MEM[13]$ 8 210002 $MEM[2] \leftarrow A$ 420000 9 = $PC \leftarrow 0$ iff. $A \neq 0$ 220000 $A \leftarrow MEM[0]$ = 010 310014 $A \leftarrow A - MEM[14]$ 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 14 220018 $A \leftarrow MEM[18]$ 15 16 2 nop 17 3 nop





git # c144985a @ 2024-10-1

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

Example (increment eler	nents in th	ne sequence	$X = \langle 0 \rangle$	$,1,2\rangle)$

		CPU
state	=	execute
PC	=	12
IR	=	420000
	=	$PC \leftarrow 0 \text{ iff. } A \neq 0$
Α	=	0
Α	=	$ PC \leftarrow 0 \text{ iff. } A \neq 0 $

MEM				
Value	Semantics			
220018	$A \leftarrow MEM[18]$			
300013	$A \leftarrow A + MEM[13]$			
210018	$MEM[18] \leftarrow A$			
220000	$A \leftarrow MEM[0]$			
300013	$A \leftarrow A + MEM[13]$			
210000	$MEM[0] \leftarrow A$			
220002	$A \leftarrow MEM[2]$			
300013	$A \leftarrow A + MEM[13]$			
210002	$MEM[2] \leftarrow A$			
220000	$A \leftarrow MEM[0]$			
310014	$A \leftarrow A - MEM[14]$			
420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$			
100000	halt			
1	nop			
220018	$A \leftarrow MEM[18]$			
1	nop			
2	nop			
3	nop			
	Value 220018 300013 210018 220000 300013 210000 220002 300013 210002 220000 310014 420000 100000 1 220018 1 2			





Notes:

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

where $\mathsf{MEM}[13\dots14]$ are constants, and X is held in $\mathsf{MEM}[15\dots17]$.

Notes:

- · The program is basically three parts
- MEM[0...2] increment the elements,
- MEM[0...2] Interinct the centerto,
 MEM[0] and MEM[0] and MEM[2], and
- MEM[9...11] control iteration of the loop body,

Example (increment elements in the sequence X = (0, 1, 2)) Value Address Semantics $A \leftarrow MEM[18]$ 220018 300013 $A \leftarrow A + MEM[13]$ 2 210018 $MEM[18] \leftarrow A$ 3 220000 $A \leftarrow MEM[0]$ 4 300013 $A \leftarrow A + MEM[13]$ CPU 5 210000 $MEM[0] \leftarrow A$ state = fetch 6 220002 $A \leftarrow MEM[2]$ PC 12 7 $A \leftarrow A + MEM[13]$ 300013 8 210002 $MEM[2] \leftarrow A$ = 100000 9 220000 $A \leftarrow MEM[0]$ 10 $A \leftarrow A - MEM[14]$ = 0 310014 11 420000 $PC \leftarrow 0 \text{ iff. } A \neq 0$ 12 100000 13 nop 14 220018 $A \leftarrow MEM[18]$ 15 nop 16 2 nop 17 3 nop



University of BRISTOL

git # c144985a @ 2024-10-1-

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

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

State = decode
PC = 13
IR = 100000
= halt
A = 0

	MEM				
Address	Value	Semantics			
0	220018	$A \leftarrow MEM[18]$			
1	300013	$A \leftarrow A + MEM[13]$			
2	210018	$MEM[18] \leftarrow A$			
3	220000	$A \leftarrow MEM[0]$			
4	300013	$A \leftarrow A + MEM[13]$			
5	210000	$MEM[0] \leftarrow A$			
6	220002	$A \leftarrow MEM[2]$			
7	300013	$A \leftarrow A + MEM[13]$			
8	210002	$MEM[2] \leftarrow A$			
9	220000	$A \leftarrow MEM[0]$			
10	310014	$A \leftarrow A - MEM[14]$			
11	420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$			
12	100000	halt			
13	1	nop			
14	220018	$A \leftarrow MEM[18]$			
15	1	nop			
16	2	nop			
17	3	nop			





Notes:

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

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

Notes:

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

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

		CPU	
-1-1-			
state	=	execute	
PC	=	13	
IR	=	100000	
	=	halt	
	_	Halt	
Α	=	0	

MEM				
Address	Value	Semantics		
0	220018	$A \leftarrow MEM[18]$		
1	300013	$A \leftarrow A + MEM[13]$		
2	210018	$MEM[18] \leftarrow A$		
3	220000	$A \leftarrow MEM[0]$		
4	300013	$A \leftarrow A + MEM[13]$		
5	210000	$MEM[0] \leftarrow A$		
6	220002	$A \leftarrow MEM[2]$		
7	300013	$A \leftarrow A + MEM[13]$		
8	210002	$MEM[2] \leftarrow A$		
9	220000	$A \leftarrow MEM[0]$		
10	310014	$A \leftarrow A - MEM[14]$		
11	420000	$PC \leftarrow 0 \text{ iff. } A \neq 0$		
12	100000	halt		
13	1	nop		
14	220018	$A \leftarrow MEM[18]$		
15	1	nop		
16	2	nop		
17	3	nop		



Conclusions

Comparison	Comparison		
A Harvard architecture	A Princeton architecture (aka. von Neumann architecture)		
has high(er) area wrt. use of 2 buses,	▶ has low(er) area wrt. use of 1 bus,		
has high(er) bandwidth wrt. use of 2 buses,	▶ has low(er) bandwidth wrt. use of 1 bus,		
implies less flexible, static memory utilisation,	implies more flexible, dynamic memory utilisation,		
can specialise instruction vs. data accesses,	cannot specialise instruction vs. data accesses,		
disallows self-modifying code.	▶ allows self-modifying code.		

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

where $\mathsf{MEM}[13 \dots 14]$ are constants, and X is held in $\mathsf{MEM}[15 \dots 17]$.

Notes:		



Conclusions

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.





Conclusions

► Take away points:

- 1. Modulo some simplifications, this really *is* how micro-processors execute programs fundamentally, there is no "magic" going on.
- 2. Both architectures are of historical, conceptual, and practical importance.
- 3. 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.

4. Variants, e.g., so-called *modified* Harvard architectures, can act as an effective compromise.

git#c144985a@2024-10-14	
Ü	
	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.
ecute programs;	
ance.	
effective compromise.	
git # c144985a @ 2024-10-14	

Notes:





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.





git # c144985a @ 2024-10-14

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:	
Notes:	
Notes:	