中国科学院大学
University of Chinese Academy of Sciences

CS101

Digital Symbol Manipulation

von Neumann Model as a Symbol Manipulation Platform

zxu@ict.ac.cn
zhangjialin@ict.ac.cn

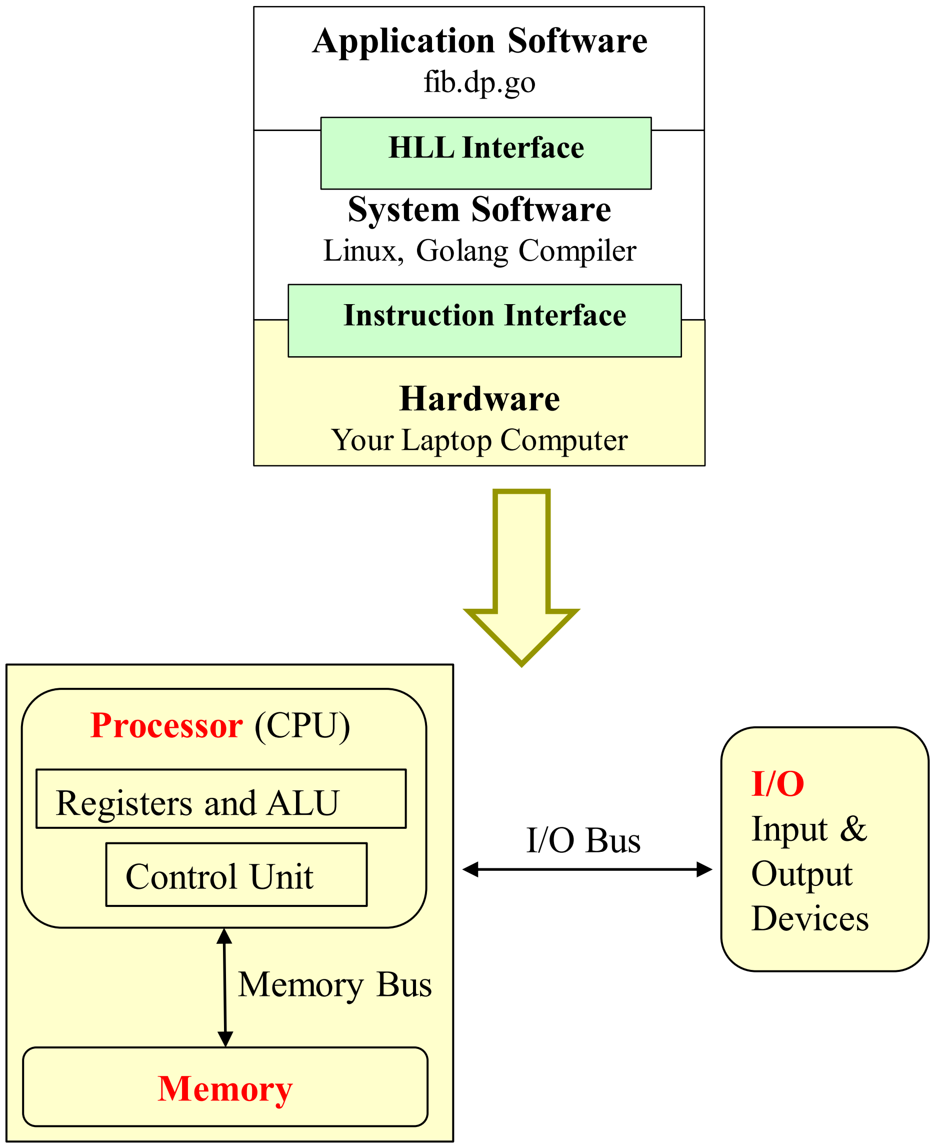
Outline

- What are digital symbol manipulations?
- Data are digital symbols
- Programs are digital symbols
- Computers are platforms of digital symbol manipulations
  - The von Neumann architecture
    - The von Neumann model of computer
  - Introducing the base-index-offset addressing mode
  - A simple computer: the Fibonacci Computer (**FC**)
    - Executing a loop
  - How does the computer work?
    - How does FC work?
    - A step-by-step walkthrough of code execution of **FC**

These slides acknowledge sources for additional data not cited in the textbook

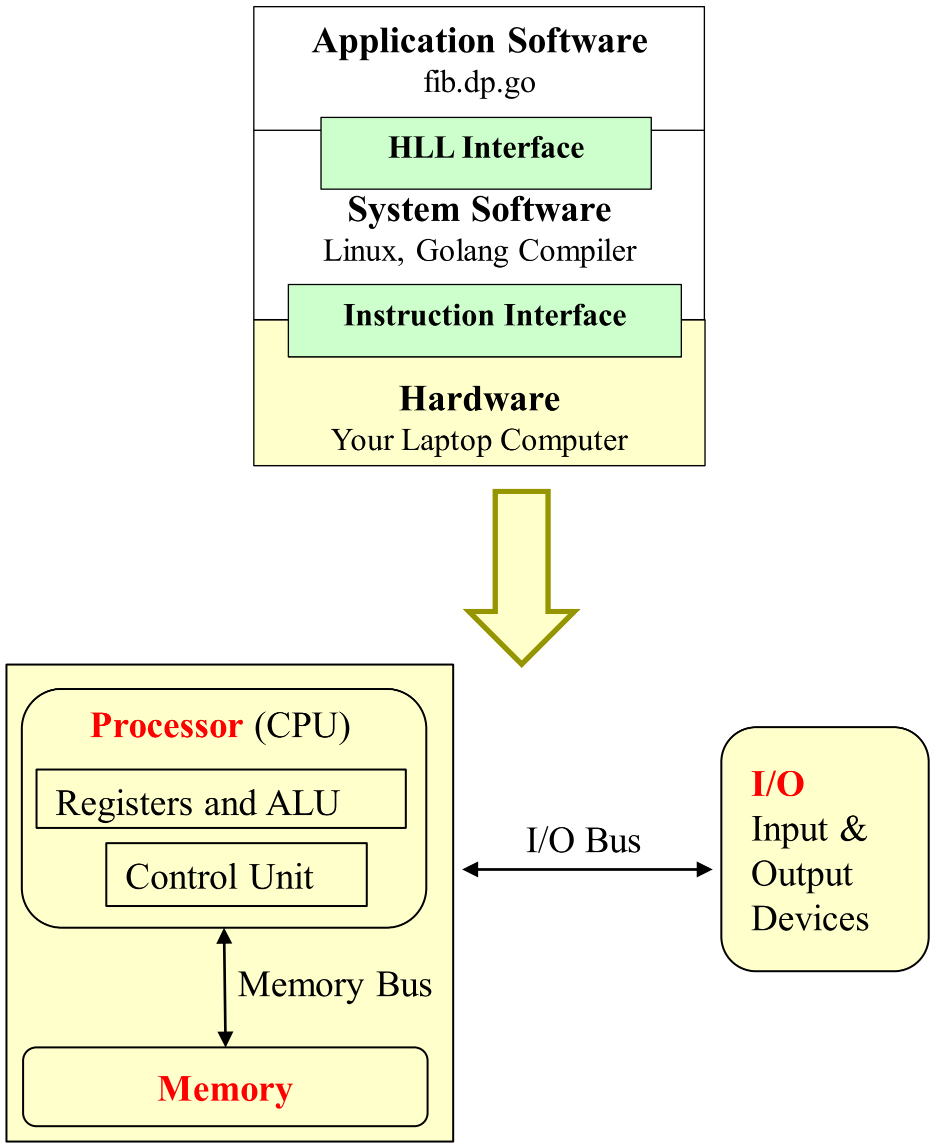
4.1 The von Neumann model of computer

- A computer has three layers
  - Hardware, system software, and application software, connected by HLL and instruction interfaces
    - HLL = High Level Language
- And five features
  - Binary: data and instructions use binary representations
  - P-M-I/O: hardware is comprised of interconnected processor, memory, and I/O devices
  - Stored program: both programs and data are stored in memory and accessed by processor
  - Instruction driven. The computer changes its state (the contents of memory and registers) only when an instruction is executed
  - Serial execution. Any program is executed by automatically executing one instruction after another



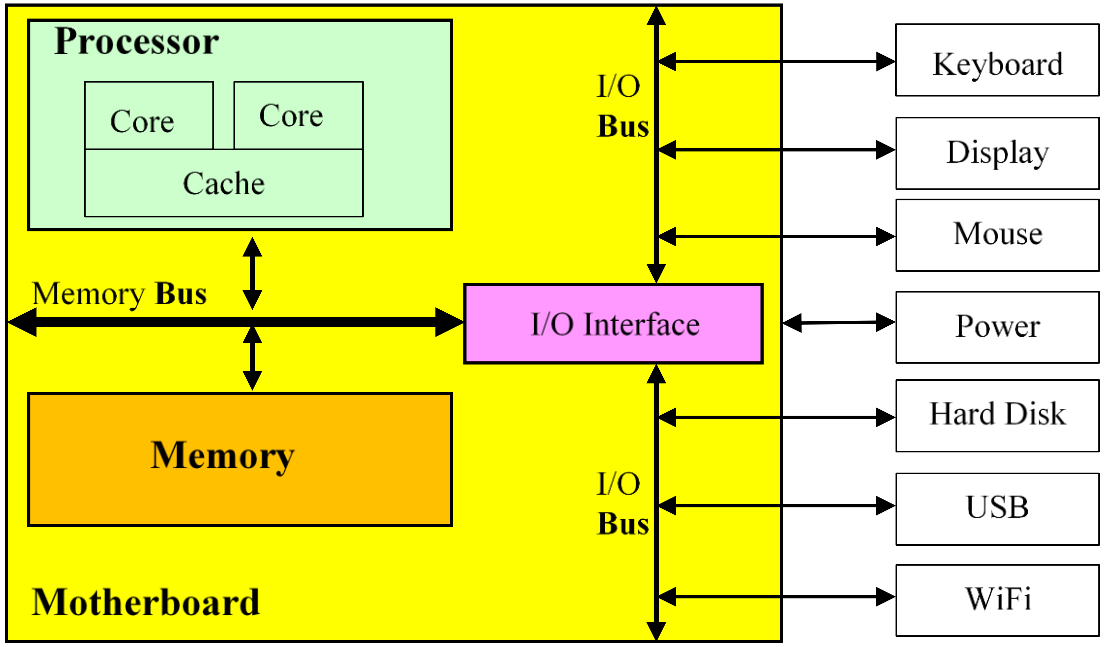
4.1 The von Neumann model of computer

- The processor is sometimes called CPU, for central processing unit
  - It executes instructions using an arithmetic logic unit (ALU) and a small number of registers, under the control of a control unit
  - A modern processor may also contain other processing units, such as graphics processing and machine learning processing
- The memory is also called main memory, accessed by processor with an instruction
  - Registers may be considered special memory cells in processor
- I/O devices include hard disk, keyboard, mouse, display, printer, networking circuitry, etc.



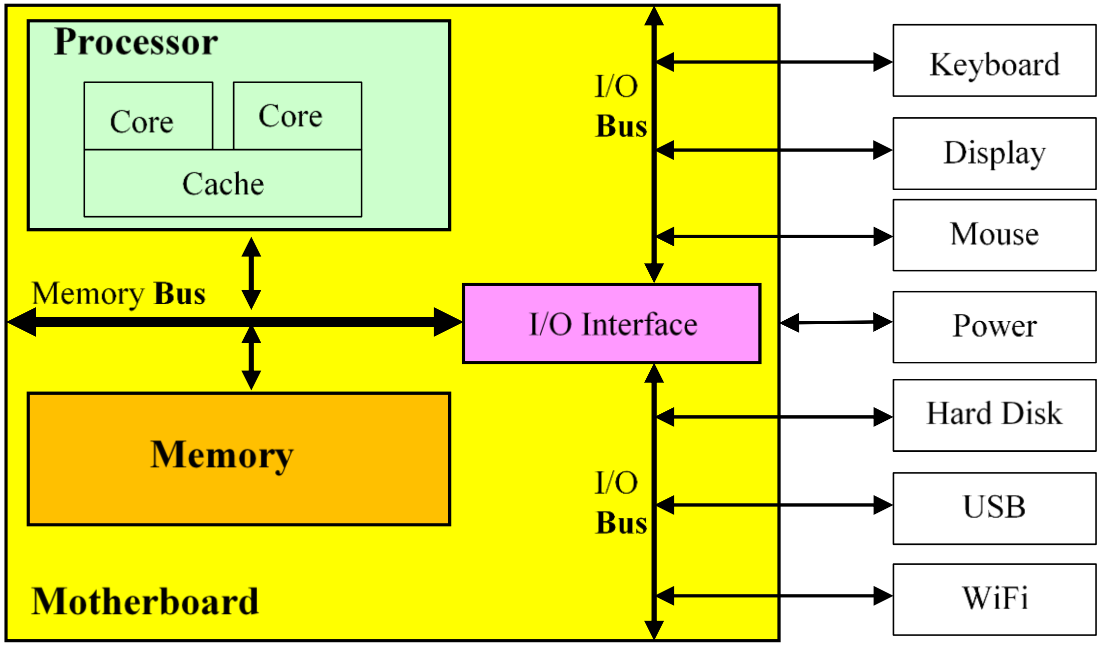
More detailed view of a computer

- A CPU is called a core in modern processors
- A processor may have multiple cores
- A processor may have a cache, which is a memory smaller but faster than the main memory
- Motherboard is the main circuit board of the computer



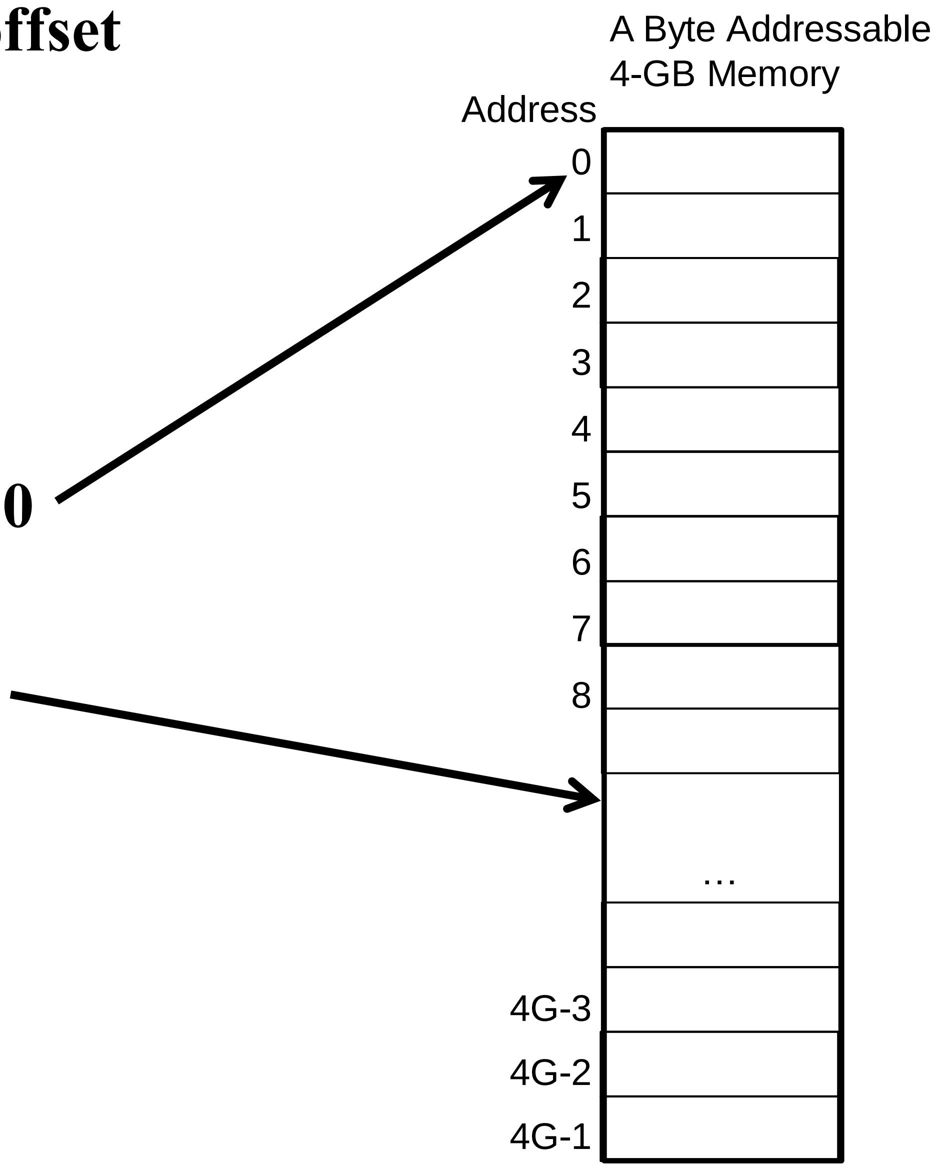
Why is the hard disk an I/O device?

- Q: A hard disk (or a USB stick) is considered an I/O device, not memory. Why? After all, they both stores information
- A: Memory is accessed directly by a single instruction, such as load/store. To access hard disk needs to execute a program (a sequence of instructions).



4.2 The base-index-offset addressing mode

- **address = base + index*8 + offset**
- Assume
  - Base = 0
  - Index = 1
- When Offset = - 8
  - **address = 0 + 1*8 + (-8) = 0**
- When Offset = 2
  - **address = 0 + 1*8 + 2 = 10**



4.3 The Fibonacci Computer (FC)

- Execute only one piece of HLL program code (the for loop)
    - Compiled to an assembly program of 12 instructions
- Byte addressable memory
    - First 12 bytes hold the 12 instructions of the code
- Registers
    - FLAGS: the status register
    - PC: the program counter, holding the address of the next instruction. Initially PC=0
    - R0: the accumulator, to hold the accumulated result. Initially R0=0
    - R1: the base register. Initially R1=12, i.e., the starting address of the fib array
    - R2: the index register
- The 12 instructions have six distinct ones
    - MOV to Register
    - MOV to Memory
    - ADD
    - INC
    - CMP
    - JL
  - The instruction set has only 6 instructions

```
fib[0] = 0                   MOV 0, R1
                             MOV R1, M[R0]            //R0=12 initially
fib[1] = 1                   MOV 1, R1
                             MOV R1, M[R0+8]
for i := 2; i < 51; i++ {    MOV 2, R2               // i:=2
   fib[i] = fib[i-1] + fib[i-2]  MOV 0, R1           // label Loop
                             ADD M[R0+R2*8-16], R1
                             ADD M[R0+R2*8-8], R1
                             MOV R1, M[R0+R2*8-0]
                             INC R2                  // i++
                             CMP 51, R2  // i < 51?
}                            JL Loop                 // if Yes, goto Loop
```

FC simplifies real cases

- Real assembly code and memory layout for an x86 processor
    - Using the AT&T assembly notations

- Here, an instruction occupies more than one byte of memory space

CPU Content		Memory Content	
Register	Value	Address	Instruction
eflags		0x672	mov $0, %rbx
rip	0x672	0x679	mov %rbx, 0(%rax)
rax	0x201010	0x67c	mov $1, %rbx
rbx	0	0x683	mov %rbx, 8(%rax)
rsi	0	0x687	mov $2, %rsi
		0x68e<for_loop>	mov $0, %rbx
		0x695	add -16(%rax, %rsi, 8), %rbx
		0x69a	add -8(%rax, %rsi, 8), %rbx
		0x69f	mov %rbx, (%rax, %rsi, 8)
		0x6a3	inc %rsi
		0x6a6	cmp $50, %rsi
		0x6aa	jl 68e
		0x201010	
		0x201018	
		0x201020	
		0x201028	

Instructions and initial configuration of FC

CPU Contents		**Memory Contents**		
Register	Value	Address	Instruction	Comments
FLAGS		0	MOV 0, R1	0→R1
PC	**0**	1	MOV R1, M[R0]	R1→M[R0]
R0	**12**	2	MOV 1, R1	1→R1
R1		3	MOV R1, M[R0+8]	R1→M[R0+8]
R2		4	MOV 2, R2	2→R2
R0: base register		5 Loop	MOV 0, R1	0→R1
Initial value=12		6	ADD M[R0+R2*8-16], R1	R1+ M[R0+R2*8-16] → R1
		7	ADD M[R0+R2*8-8], R1	R1+ M[R0+R2*8-8] → R1
R1: accumulator		8	MOV R1, M[R0+R2*8-0]	R1→ M[R0+R2*8-0]
R2: index register		9	INC R2	R2+1→R2
		10	CMP 51, R2	Compare R2 to 51, status→FLAGS
Address=base+ index*8+offset		11	JL Loop	If FLAGS is "<", Loop→PC
		12		fib[0]
fib[i-2]'s address		20		fib[1]
=R0+R2*8 -16		28		fib[2]
		36		fib[3]
fib[0]'s address		……		……
=12+2*8-16=12		412		fib[50]

Instructions and initial configuration of FC

CPU Contents		**Memory Contents**		
Register	Value	Address	Instruction	Comments
FLAGS		0	MOV 0, R1	0→R1
PC	**0**	1	MOV R1, M[R0]	R1→M[R0] fig[0] =0
R0	**12**	2	MOV 1, R1	1→R1
R1		3	MOV R1, M[R0+8]	R1→M[R0+8] fig[1] =1
R2		4	MOV 2, R2	2→R2 i:=2
R0: base register		5 Loop	MOV 0, R1	0→R1
Initial value=12		6	ADD M[R0+R2*8-16], R1	R1+ M[R0+R2*8-16] → R1
		7	ADD M[R0+R2*8-8], R1	R1+ M[R0+R2*8-8] → R1
R1: accumulator		8	MOV R1, M[R0+R2*8-0]	R1→ M[R0+R2*8-0]
R2: index register		9	INC R2	R2+1→R2
		10	CMP 51, R2	Compare R2 to 51, status→FLAGS
Address=base+ index*8+offset		11	JL Loop	If FLAGS is "<", Loop→PC
		12		fib[0] **M[12]: memory cell**
fib[i-2]'s address		20		fib[1] **when address=12**
=R0+R2*8 -16		28		fib[2]
		36		fib[3] Each integer holds 8 bytes
fib[0]'s address		……		……
=12+2*8-16=12		412		fib[50]

4.4 A step-by-step walkthrough

- Notes on the five common features of the von Neumann model
  - Binary: data and instructions use binary representations
    - Check. Although we humans work in decimal in walkthrough
  - P-M-I/O: hardware is comprised of processor, memory, and I/O devices
    - Check. Although we ignore I/O in this example
  - Stored program: both programs and data are stored in memory and accessed by processor
    - Check. Program in addresses 0~11, and data in addresses 12~419
  - Instruction driven. The computer changes its state (the contents of memory and registers) only when an instruction is executed
    - Check. Ensure this in walkthrough.
  - Serial execution. Any program is executed by automatically executing one instruction after another
    - Check. Ensure this in walkthrough
    - Each step should have changes in two places: PC and register/memory

Step 1

CPU Contents		Memory Contents	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	**1**	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	**0**	3	MOV R1, M[R0+8]
R2		4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	
		20	
		28	
		36	

Step 2

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	**2**	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	0	3	MOV R1, M[R0+8]
R2		4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	**0**
		20	
		28	
		36	

Step 3

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	**3**	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	**1**	3	MOV R1, M[R0+8]
R2		4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	
		28	
		36	

Step 4

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	**4**	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	1	3	MOV R1, M[R0+8]
R2		4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	**1**
		28	
		36	

Step 5

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	5	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	1	3	MOV R1, M[R0+8]
R2	2	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	
		36	

Step 6

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	6	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	0	3	MOV R1, M[R0+8]
R2	2	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	
		36	

Step 7

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	7	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	0	3	MOV R1, M[R0+8]
R2	2	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	
		36	

Step 8

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	8	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	1	3	MOV R1, M[R0+8]
R2	2	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	
		36	

Step 5

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	5	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	1	3	MOV R1, M[R0+8]
R2	2	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	
		36	

Step 6

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	6	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	0	3	MOV R1, M[R0+8]
R2	2	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	
		36	

Step 7

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	7	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	0	3	MOV R1, M[R0+8]
R2	2	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	
		36	

Add M[12], R1
M[12]+R1→R1

Step 8

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	8	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	1	3	MOV R1, M[R0+8]
R2	2	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	
		36	

fig[i]=fig[i-1]+fig[i-2]
when i==2
fig[i-2]'s address =
  R0+R2*8-16=12+2*8-16=12
fig[i-1]'s address =
  R0+R2*8-8=12+2*8-8=20
fig[i]'s address =
  base+index*8+offset
  R0+R2*8-0=12+2*8=28

Step 5

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	5	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	1	3	MOV R1, M[R0+8]
R2	2	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	
		36	

Step 6

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	6	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	0	3	MOV R1, M[R0+8]
R2	2	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	
		36	

Step 7

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	7	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	0	3	MOV R1, M[R0+8]
R2	2	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	
		36	

Add M[12], R1
M[12]+R1 → R1
0+0 → R1

Step 8

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	8	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	1	3	MOV R1, M[R0+8]
R2	2	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	
		36	

fig[i]=fig[i-1]+fig[i-2]
when i==2
fig[i-2]'s address =
R0+R2*8-16=12+2*8-16=12
fig[i-1]'s address =
R0+R2*8-8=12+2*8-8=20
fig[i]'s address =
base+index*8+offset
R0+R2*8-0=12+2*8=28

Step 5

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	5	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	1	3	MOV R1, M[R0+8]
R2	2	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	
		36	

Step 6

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	6	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	0	3	MOV R1, M[R0+8]
R2	2	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	
		36	

Step 7

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	7	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	0	3	MOV R1, M[R0+8]
R2	2	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	
		36	

Add M[12], R1
M[12]+R1→R1
0+0 → R1

fig[i]=fig[i-1]+fig[i-2]
when i==2
fig[i-2]'s address =
R0+R2*8-16=12+2*8-16=12
fig[i-1]'s address =
R0+R2*8-8=12+2*8-8=20
fig[i]'s address =
base+index*8+offset
R0+R2*8-0=12+2*8=28

Step 8

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	8	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	1	3	MOV R1, M[R0+8]
R2	2	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	
		36	

Add M[20], R1
M[20]+R1→R1

Step 5

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	5	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	1	3	MOV R1, M[R0+8]
R2	2	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	
		36	

Step 6

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	6	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	0	3	MOV R1, M[R0+8]
R2	2	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	
		36	

Step 7

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	7	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	0	3	MOV R1, M[R0+8]
R2	2	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	
		36	

Add M[12], R1
M[12]+R1→R1
0+0 → R1

Step 8

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	8	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	1	3	MOV R1, M[R0+8]
R2	2	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	
		36	

fig[i]=fig[i-1]+fig[i-2]
when i==2
fig[i-2]'s address =
  R0+R2*8-16=12+2*8-16=12
fig[i-1]'s address =
  R0+R2*8-8=12+2*8-8=20
fig[i]'s address =
  base+index*8+offset
  R0+R2*8-0=12+2*8=28

Add M[20], R1
M[20]+R1→R1
1 + 0 → R1

Step 9

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	**9**	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	1	3	MOV R1, M[R0+8]
R2	2	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	**1**
		36	

Step 10

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS		0	MOV 0, R1
PC	**10**	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	1	3	MOV R1, M[R0+8]
R2	**3**	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	1
		36	

Step 11

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS	<	0	MOV 0, R1
PC	**11**	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	1	3	MOV R1, M[R0+8]
R2	3	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	1
		36	

Step 12

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS	<	0	MOV 0, R1
PC	**5**	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	1	3	MOV R1, M[R0+8]
R2	3	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	1
		36	

Step 13

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS	<	0	MOV 0, R1
PC	6	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	0	3	MOV R1, M[R0+8]
R2	3	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	1
		36	

Step 14

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS	<	0	MOV 0, R1
PC	7	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	1	3	MOV R1, M[R0+8]
R2	3	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	1
		36	

Step 15

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS	<	0	MOV 0, R1
PC	8	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	2	3	MOV R1, M[R0+8]
R2	3	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	1
		36	

Step 16

CPU Content		Memory Content	
Register	Value	Address	Instruction
FLAGS	<	0	MOV 0, R1
PC	9	1	MOV R1, M[R0]
R0	12	2	MOV 1, R1
R1	2	3	MOV R1, M[R0+8]
R2	3	4	MOV 2, R2
		5 Loop	MOV 0, R1
		6	ADD M[R0+R2*8-16], R1
		7	ADD M[R0+R2*8-8], R1
		8	MOV R1, M[R0+R2*8-0]
		9	INC R2
		10	CMP 51, R2
		11	JL Loop
		12	0
		20	1
		28	1
		36	2