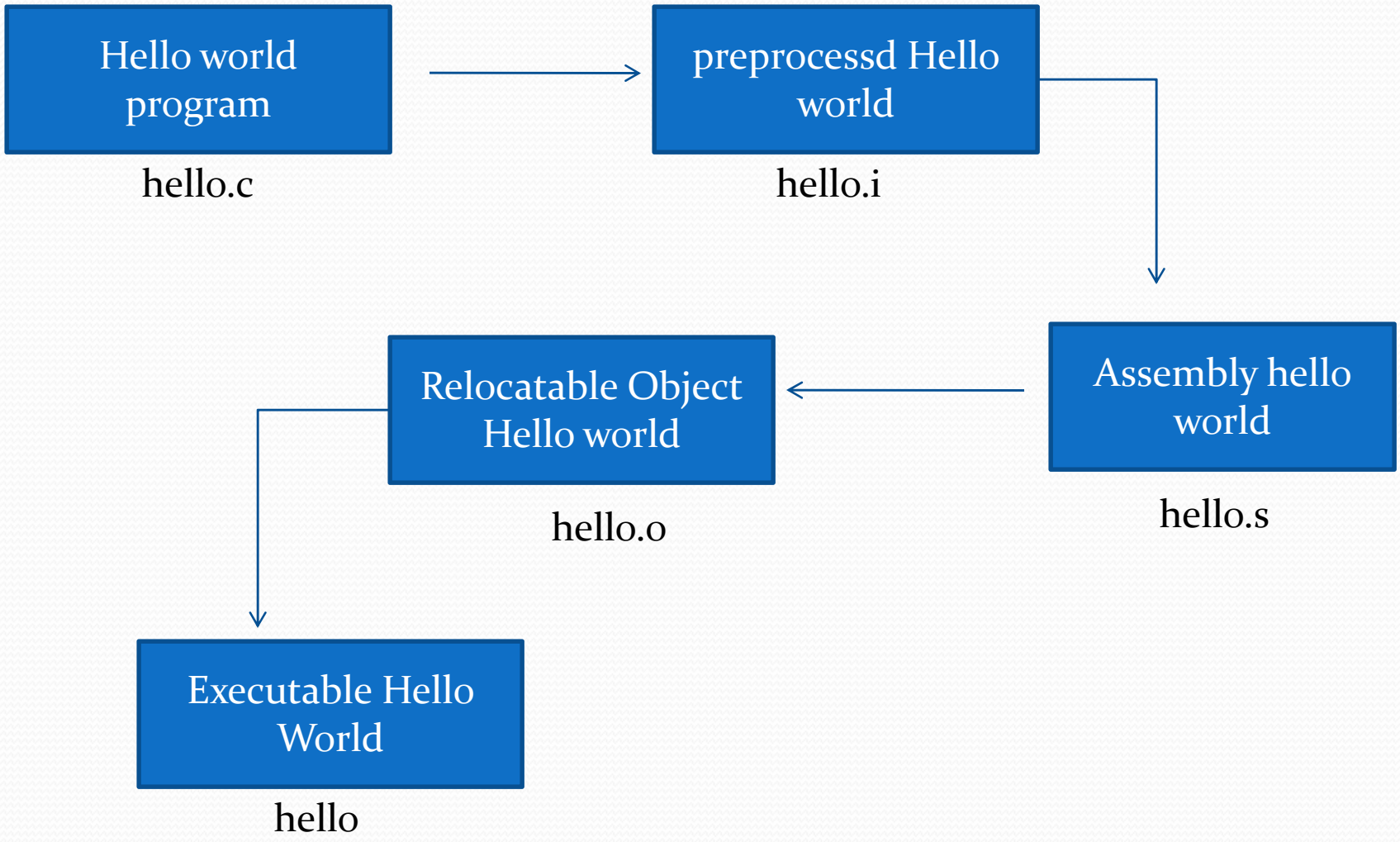
Assembler Fundamentals

15-123

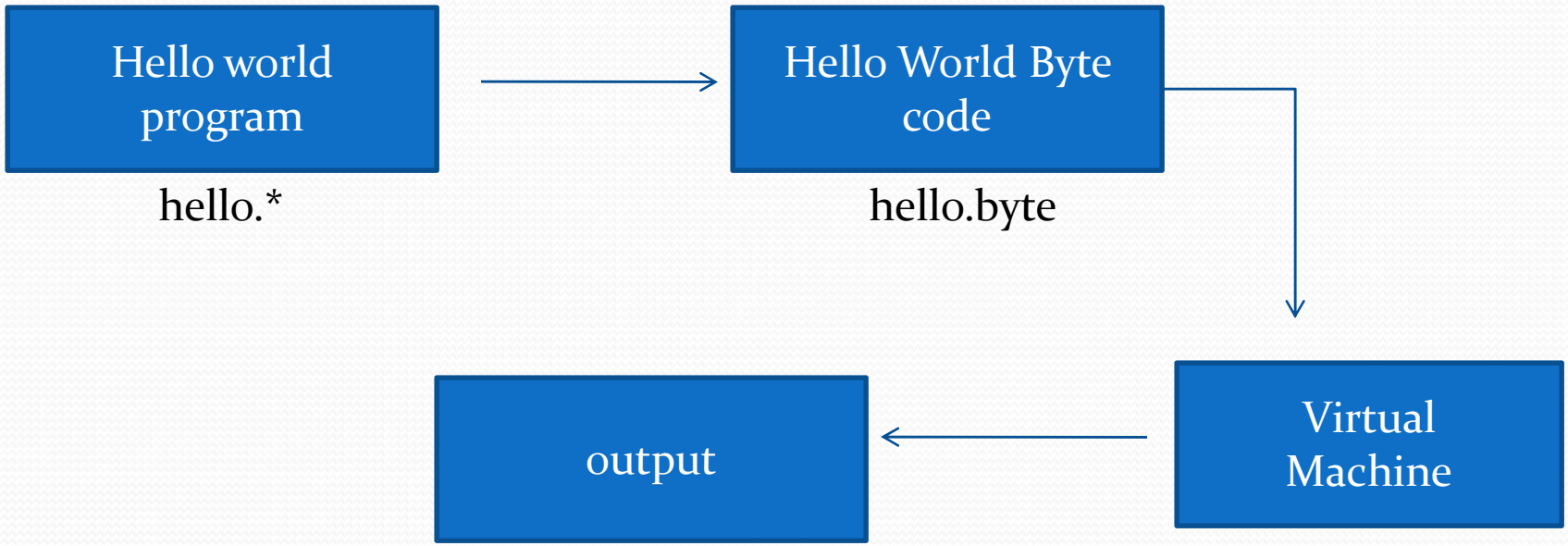
Systems Skills in C and Unix

Some background Stuff

Programs are Translated into binaries



Programs are Translated into byte code



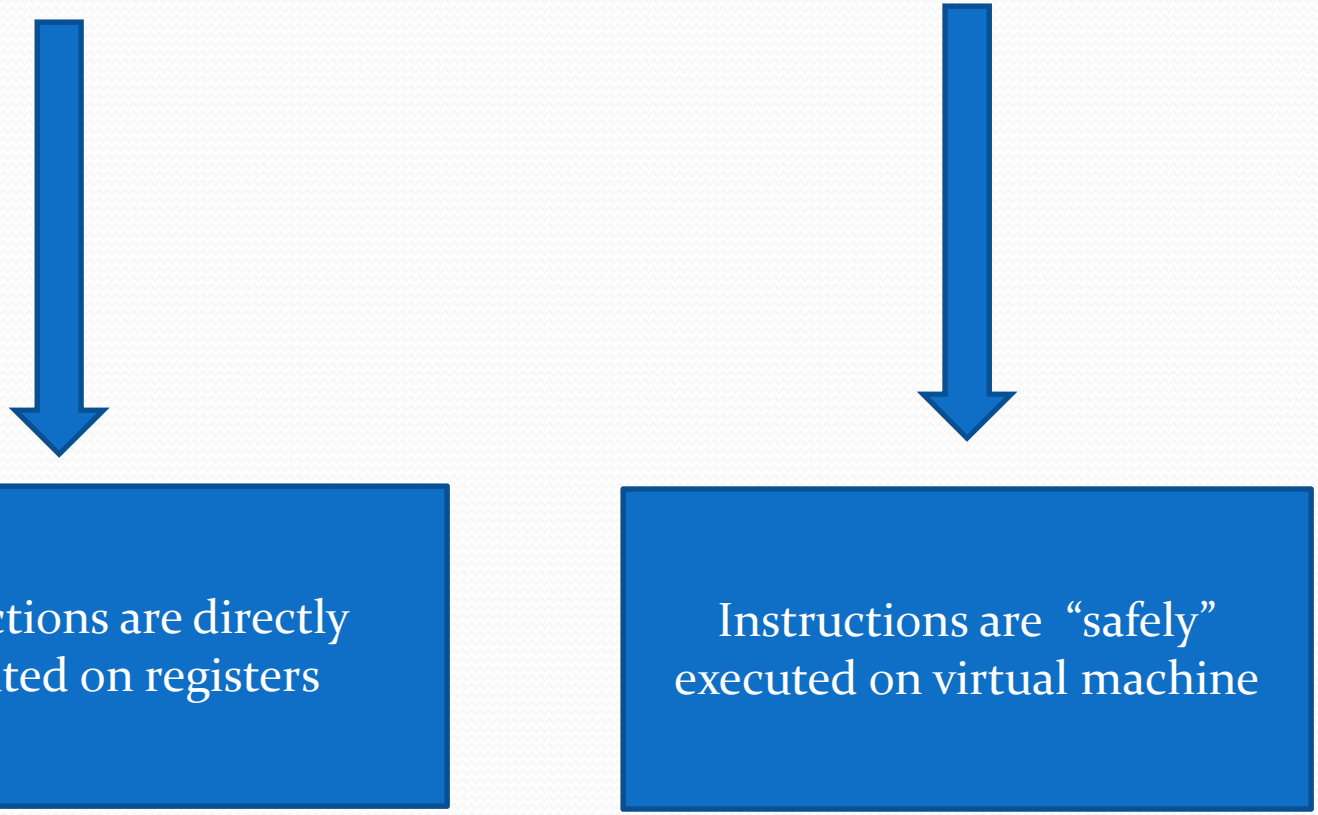
Demo code

Byte code versus assembly code

```
#<main>
00 00              # number of arguments = 0
00 00              # number of local variables = 0
00 06              # code length = 6 bytes
10 19     # bipush 25        # 25
10 24     # bipush 36        # 36
60        # iadd             # (25 + 36)
B0        # return           #
```

```
movl    $25, -8(%rbp)
movl    $36, -4(%rbp)
movl    -4(%rbp), %edx
movl    -8(%rbp), %eax
addl    %edx, %eax
leave
ret
```

Register based vs stack based

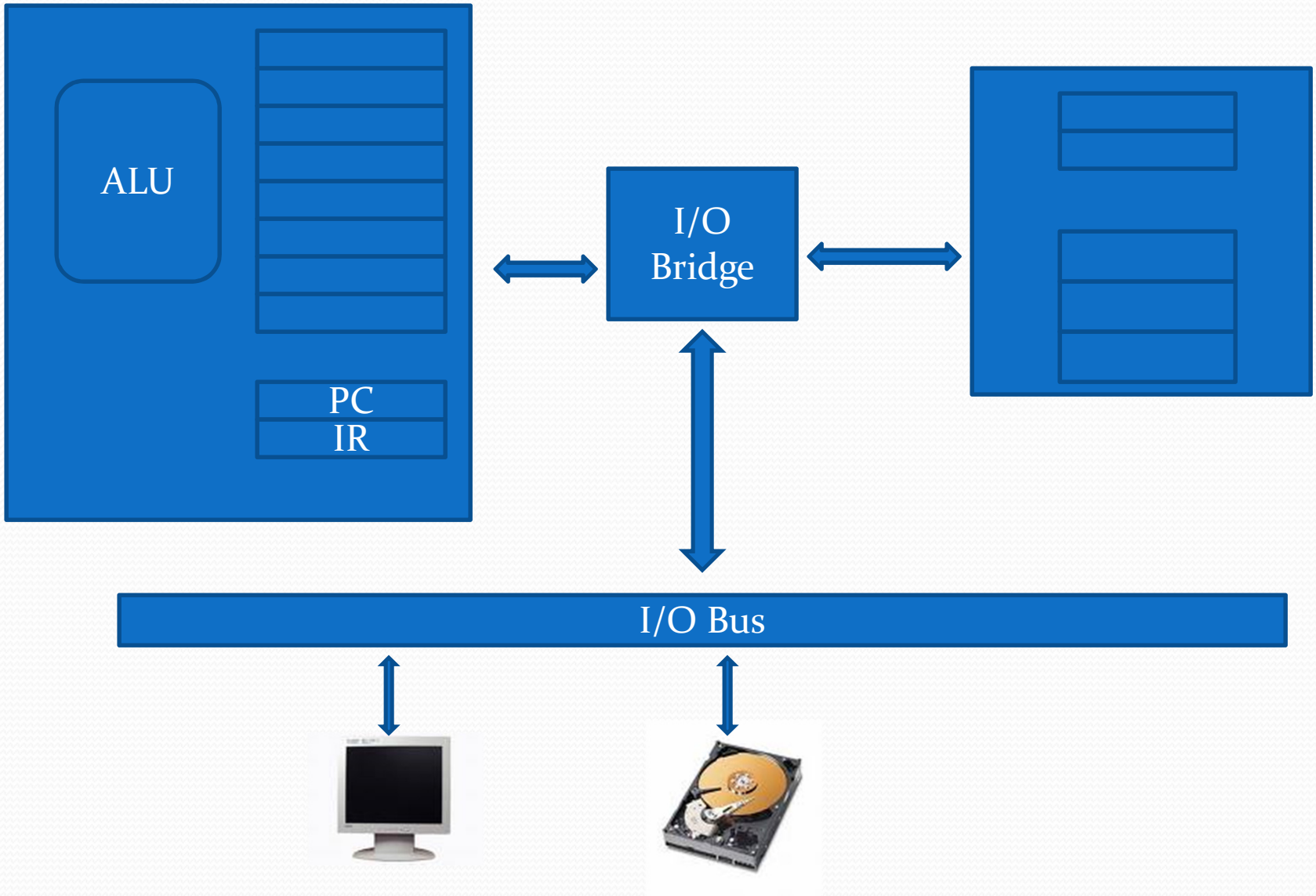


Functions as data

- Virtual machines (VM) treat programs as data
- Program is a file of bytes
- VM has a stack based model of implementation
- Each function call uses its own stack to execute the instructions
- The machine hardware only executes the VM instructions
- No program instructions get executed by hardware

Some other details

All programs get executed



Cost of computing is important

- Moving data is expensive
- Large storage devices are slower than smaller storage devices (hard drive vs RAM)
  - Capacity: RAM/hard drive = 1/100
  - Access : RAM/Hard drive = 1/10,000,000
- RAM versus Registers
  - Capacity: Register/RAM = 32 bits/2 GB
  - Access: Register :RAM access = 100:1

Speeding up with Cache can help

- Cache memories
  - Smaller faster storage devices
  - Stores data that the processor is likely to need in the near future
  - Cache memory is directly connected through bus interface
  - Goal is to make cache memory access as fast as register access

Understanding Instruction Set Architecture

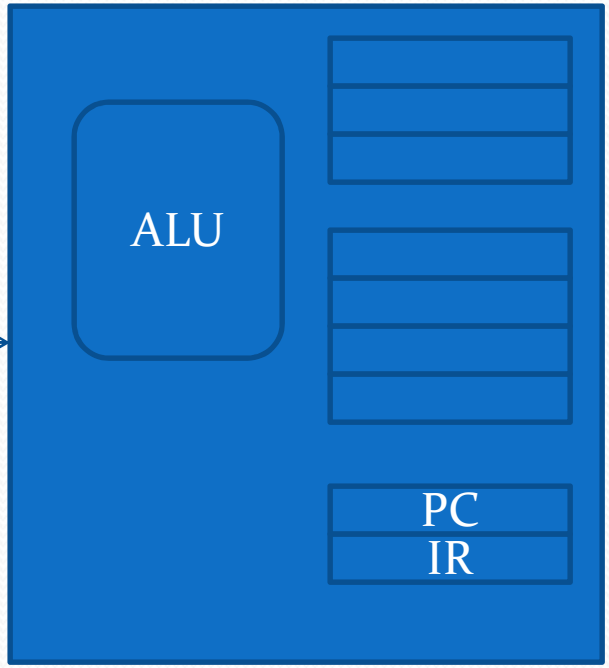
Instruction set architecture

- provides a perspective of the processor from assembly language or machine language programmer's point of view.

```
0 LOADI A       1
1 LOADI B       48
2 IN            C       0
3 SUB           D       C       B
4 IN            C       0
5 SUB           E       C       B

6 LTE   D       E
7 NOT
8 CJMP  (Line 13)
9 ADD           C       D       B
10 OUT          C       15
11 ADD          D       D       A

13 HLT
```



Instruction set architecture

- ISA describes the *instructions that processor understands*, including register set and how the memory is organized.
- A real world processor ISA would include few additional items such as data types; interrupt handlers, exception handling etc. *ISA is part of the computer architecture specific to a particular hardware.*

Registers

- Registers are special purpose memory locations
- Most assembly instructions directly operate on registers
  - loading data into registers from memory
  - performing operations on data and storing data back in memory
- The registers are named like
  - eax, ebx, ecx
  - ebp and esp - for manipulating the base pointer and stack pointer
- The size of a register (say 32-bit) and number of registers (say 8) depends on particular computer architecture.
- A typical register instruction in assembly
  - **movl $10, %eax**

A Hypothetical Machine

Register	Number	Notes
Z	000	Constant: Always zero (0)
A	001	
B	010	
C	011	
D	100	
E	101	
F	110	
G	111	
PC		Program Counter. 24 bits wide. Not addressable
IR		Instruction Register. 32 bits wide. Not addressable.

Question: How many addressable units are in our memory model? Answer based on PC

Basic Instructions

- The basic instructions for a computer are
  - *branch instructions*
    - *jmp*
  - *I/O instructions*
    - *Load and save*
  - *Arithmetic instructions*
    - *add, mul*
  - *Device instructions*
    - *Read, write*
  - *comparison instructions*
    - *If (x > y)*

Instructions

CONTROL INSTRUCTIONS

Instruction	Op	Address	Function
HLT	0000	XXXX XXXX XXXX XXXX XXXX XXXX	Stop simulation
JMP	0001	0000 AAAA AAAA AAAA AAAA AAAA	Jump (line number)
CJMP	0010	0000 AAAA AAAA AAAA AAAA AAAA	Jump if true
OJMP	0011	0000 AAAA AAAA AAAA AAAA AAAA	Jump if overflow

LOAD-STORE INSTRUCTIONS

Instruction	Op	Register	Value	Function
LOAD	0100	0RRR	AAAA AAAA AAAA AAAA AAAA AAAA	Load (hex address)
STORE	0101	0RRR	AAAA AAAA AAAA AAAA AAAA AAAA	Store (hex address)
LOADI	0110	0RRR	0000 0000 IIII IIII IIII IIII	Load Immediate
NOP	0111	0000	0000 0000 0000 0000 0000 0000	no operation

MATH INSTRUCTIONS

Instruction	Op	Reg0	Reg1	Reg2		Function
ADD	1000	0RRR	0RRR	0RRR	0000 0000 0000 0000	Reg0 = (Reg1 + Reg2)
SUB	1001	0RRR	0RRR	0RRR	0000 0000 0000 0000	Reg0 = (Reg1 - Reg2)

DEVICE I/O

Instruction	-Op-	Reg0	0000 0000 0000 0000	Port	Function
IN	1010	0RRR	0000 0000 0000 0000	PPPP PPPP	Read Port into Reg0
OUT	1011	0RRR	0000 0000 0000 0000	PPPP PPPP	Write Reg0 out to Port

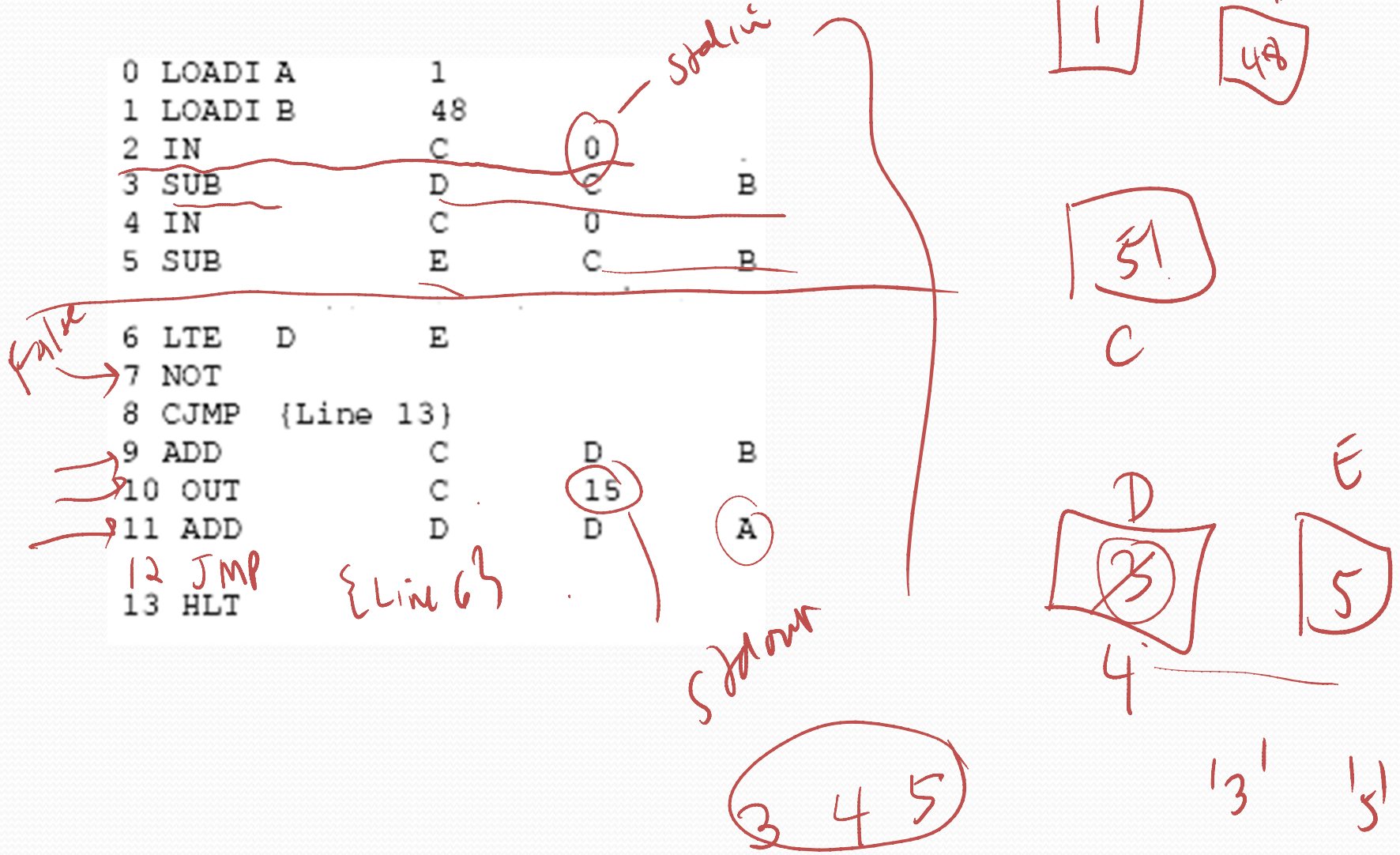
COMPARISON

Instruction	-Op-	Reg0	Reg1		Function
EQU	1100	0RRR	0RRR	0000 0000 0000 0000 0000	Cflg = (Reg0 == Reg1)
LT	1101	0RRR	0RRR	0000 0000 0000 0000 0000	Cflg = (Reg0 < Reg1)
LTE	1110	0RRR	0RRR	0000 0000 0000 0000 0000	Cflg = (Reg0 <= Reg1)
NOT	1111	0000	0000	0000 0000 0000 0000 0000	Cflg = (!Cflg)

Sample program

```
0  LOADI A        1
1  LOADI B        48
2  IN             C        0
3  SUB            D        C        B
4  IN             C        0
5  SUB            E        C        B

6  LTE    D       E
7  NOT
8  CJMP   {Line 13}
9  ADD            C        D        B
10 OUT            C        15
11 ADD            D        D        A
12 JMP
13 HLT
```



From assembly to object code

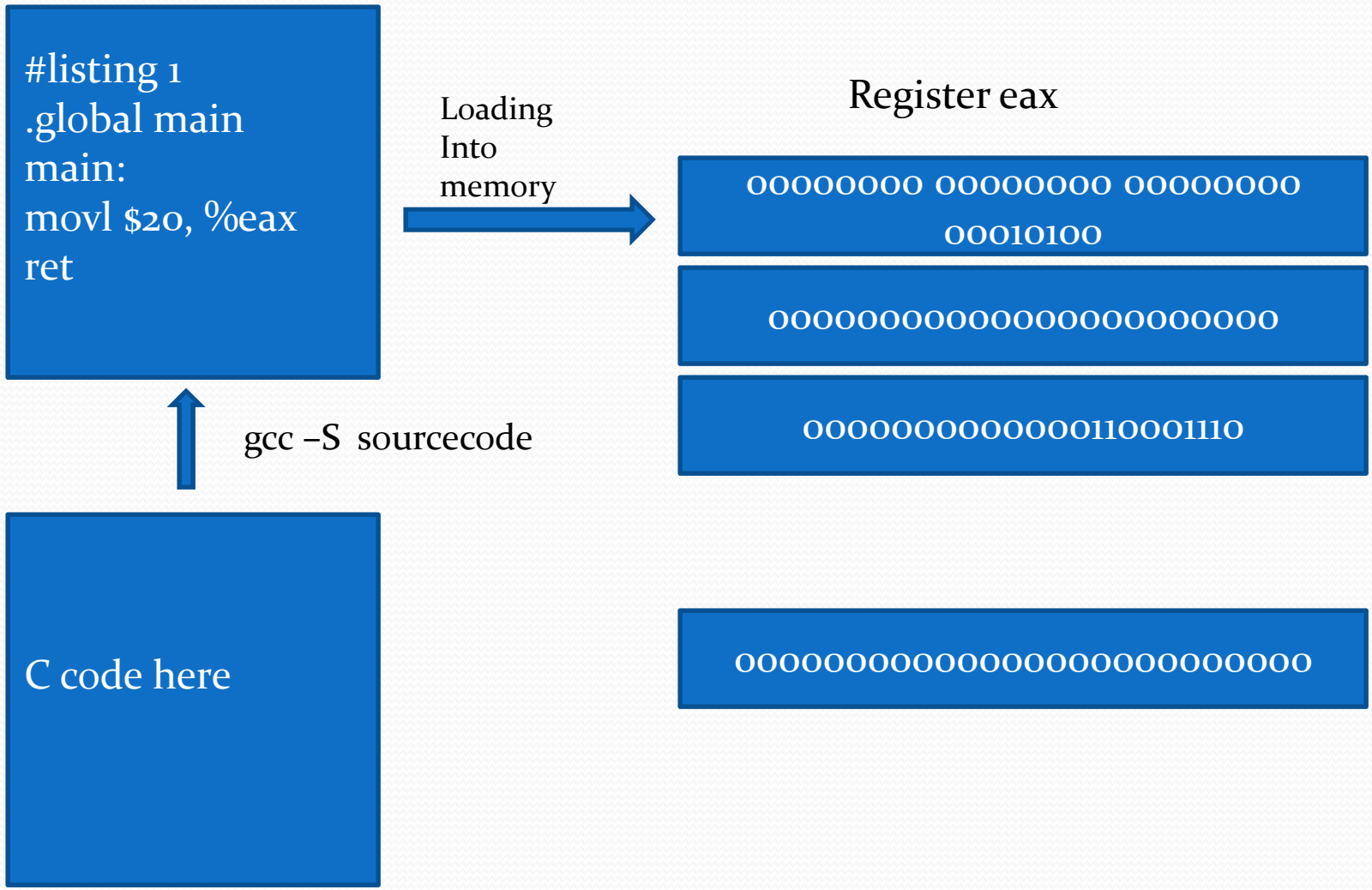
```
0 LOADI A       1
1 LOADI B       48
2 IN            C       0
3 SUB           D       C       B
4 IN            C       0
5 SUB           E       C       B

6 LTE   D       E
7 NOT
8 CJMP  {Line 13}
9 ADD           C       D       B
10 OUT          C       15
11 ADD          D       D       A

13 HLT
```

1. 0x61000001
2. 0x62000030
3. 0xA3000000
4. 0x94320000
5. 0xA3000000
6. 0x95320000
7. 0xE4500000
8. 0xF0000000
9. 0x20000034
10. 0x83420000
11. 0xB300000F
12. 0x84410000
13. 0x10000018
14. 0x00000000

instructions gets loaded into registers



Exercises
class activity

Exercise 1

- Write a program to add the numbers 2 and 5 and output to port #15 (output port). Then convert to machine code.

Exercise 2

- Write a program that reads a single digit integer from keyboard and output.

Exercise 3

- Write a program that reads a single digit integer from keyboard and output the number if the number is greater or equal to 5.

Exercise 4

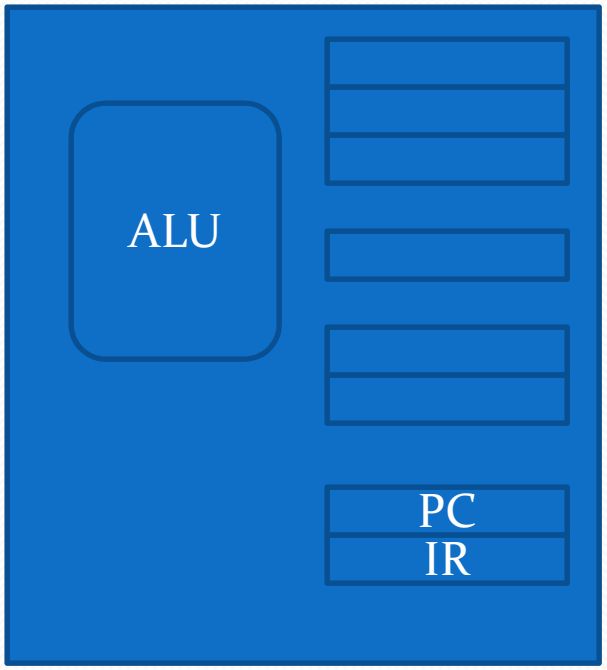
- Write a program that reads a single digit integer from keyboard and output all numbers between 1 and number

Simulation

How to think about a simulator

1. 0x61000001
2. 0x62000030
3. 0xA3000000
4. 0x94320000
5. 0xA3000000
6. 0x95320000
7. 0xE4500000
8. 0xF0000000
9. 0x20000034
10. 0x83420000
11. 0xB300000F
12. 0x84410000
13. 0x10000018
14. 0x00000000

Instructions in RAM



Simulator hardware Components

- An array of eight 32-bit registers
- A 32-bit register for IR and a 24-bit register for PC
- An array of bytes to simulate RAM. Each instruction takes 32-bytes
  - How much memory is needed based on PC?

Object Code Parser

- Load the instruction to IR
  - 0x61000001
- Parse the instruction using bit masks to extract the meaning

  0x61000001
- Execute the instruction → LOADI A 1
  - Load 1 to register A

Code and Data

Code and Data are inseparable

- A 160 character C program that computes the first 800 digits of pi.

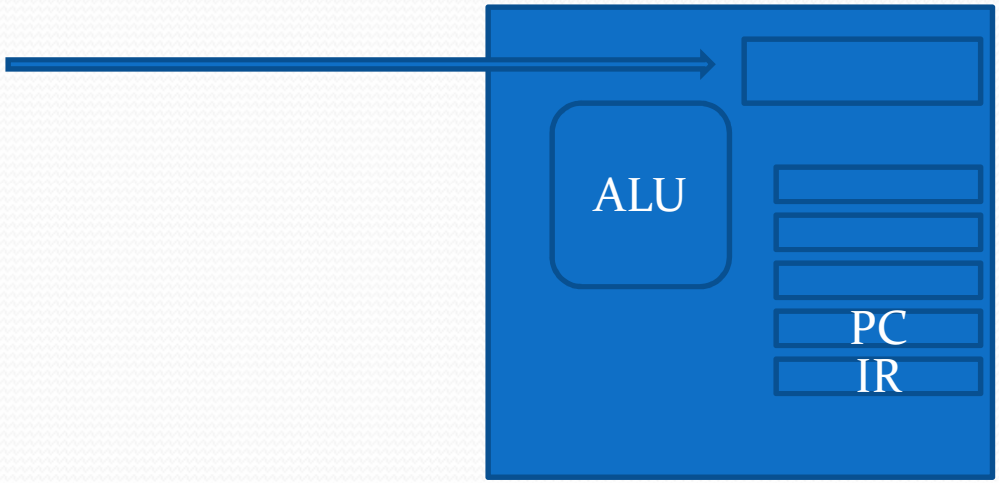
```
int a=10000,b,c=2800,d,e,f[2801],g;main(){for(;b-
c;)f[b++]=a/5; for(;d=0,g=c*2;c-
14,printf("%.4d",e+d/a),e=d%a)for(b=c;d+=f[b]*a, f[b]=d%-
-g,d/=g--,--b;d*=b);}
```

Loading code/data from memory

1. 0x61000001
2. 0x62000030
3. 0xA3000000
4. 0x94320000
5. 0xA3000000
6. 0x95320000
7. 0xE4500000
8. 0xF0000000
9. 0x20000034
10. 0x83420000
11. 0xB300000F
12. 0x84410000
13. 0x10000018
14. 0x00000000

14. 0x00000001
15. 0x00010010
16. 0x00010011
17. 0x01001001
18. 0x10010010

LOAD Register hex-address

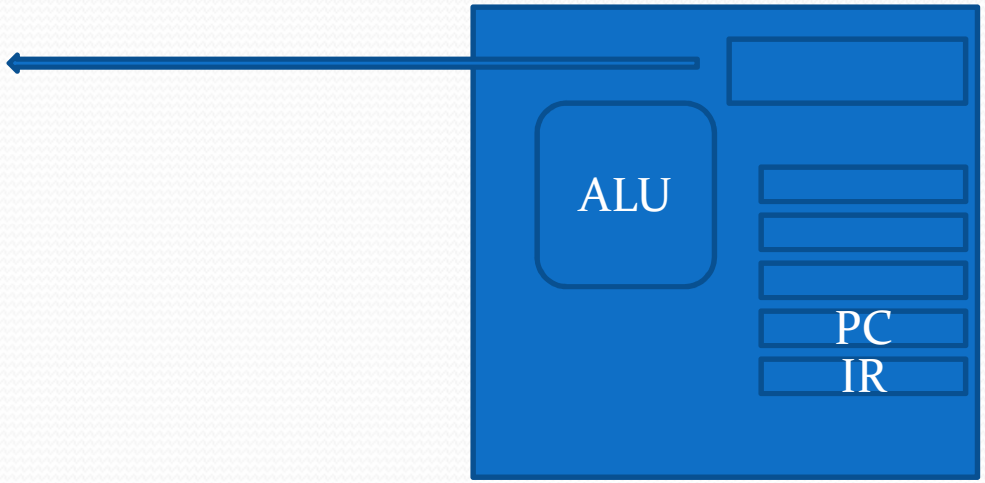


Storing data in memory

1. 0x61000001
2. 0x62000030
3. 0xA3000000
4. 0x94320000
5. 0xA3000000
6. 0x95320000
7. 0xE4500000
8. 0xF0000000
9. 0x20000034
10. 0x83420000
11. 0xB300000F
12. 0x84410000
13. 0x10000018
14. 0x00000000

STORE Register hex-address

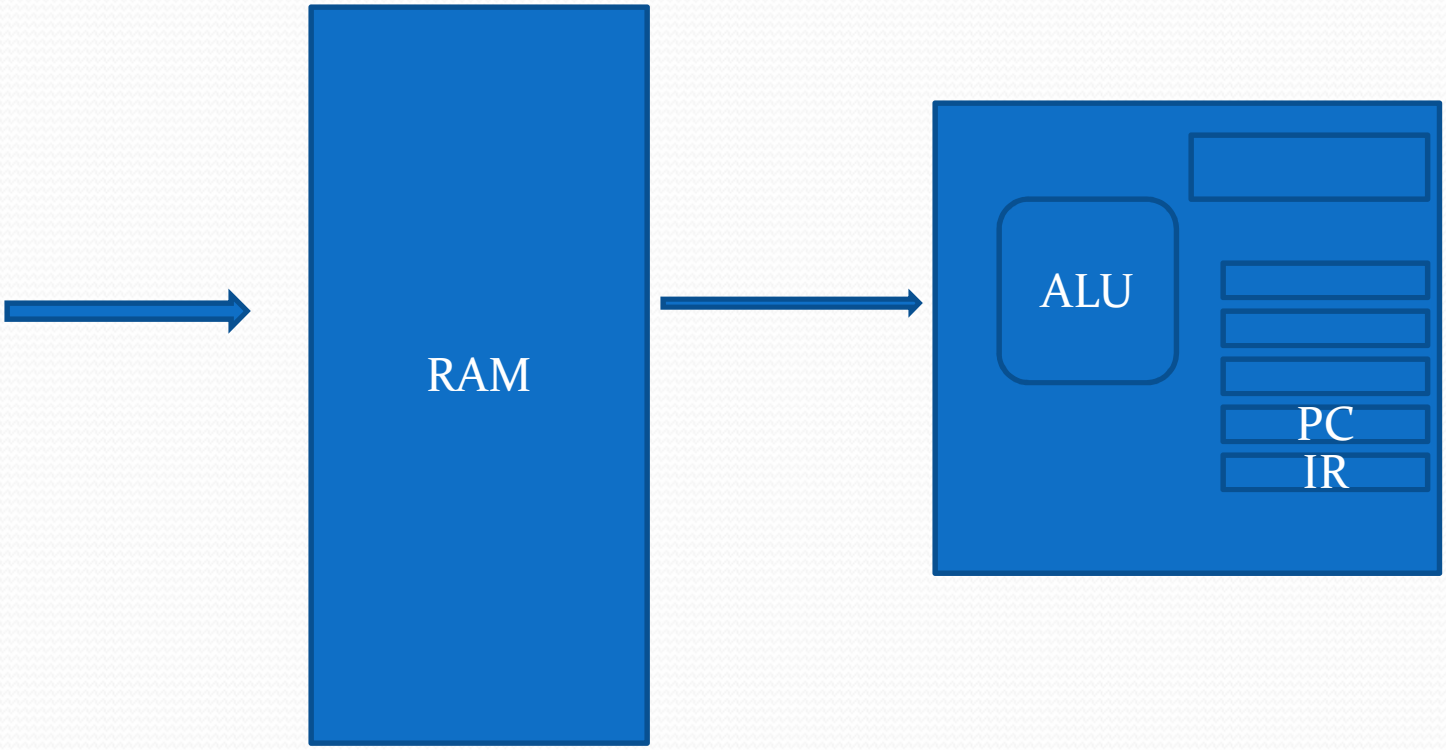
14. 0x00000001
15. 0x00010010
16. 0x00010011
17. 0x01001001
18. 0x10010010



Code and Data

1. 0x61000001
2. 0x62000030
3. 0xA3000000
4. 0x94320000
5. 0xA3000000
6. 0x95320000
7. 0xE4500000
8. 0xF0000000
9. 0x20000034
10. 0x83420000
11. 0xB300000F
12. 0x84410000
13. 0x10000018
14. 0x00000000

14. 0x00000001
15. 0x00010010
16. 0x00010011
17. 0x01001001
18. 0x10010010



Coding Examples