Hacettepe University
Computer Engineering Department

Programming in HMMM

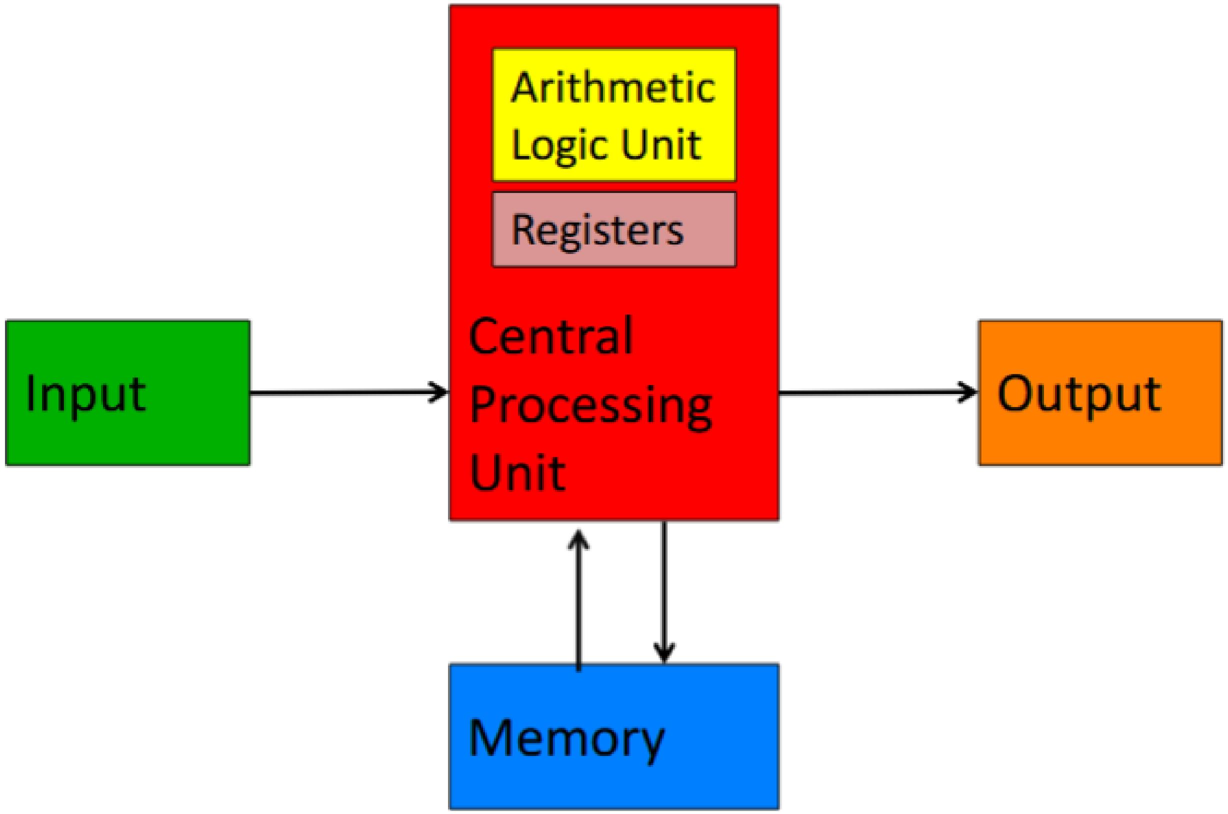
BBM103 Introduction to Programming Lab 1
Week 3

Fall 2017

Von Neumann Architecture

- A **program** (a list of instructions) is stored in the main memory.
  - **Stored Program Concept**

- Instructions are copied (one at a time) into the **instruction register** in the CPU for execution.



Von Neumann Architecture

<u>processing</u>

CPU

central processing unit ***registers***

r1 General-purpose register, r1

r2 General-purpose register, r2

Von Neumann bottleneck

<u>program</u>

RAM

random access memory locations

0	0000 0001 0000 0001
1	1000 0010 0001 0001
2	0110 0010 0010 0001
3	0000 0010 0000 0010
4	0000 0000 0000 0000
5	
6	

(all bits)

Programs are stored in memory in ***machine language***.

The Power of the Stored Program

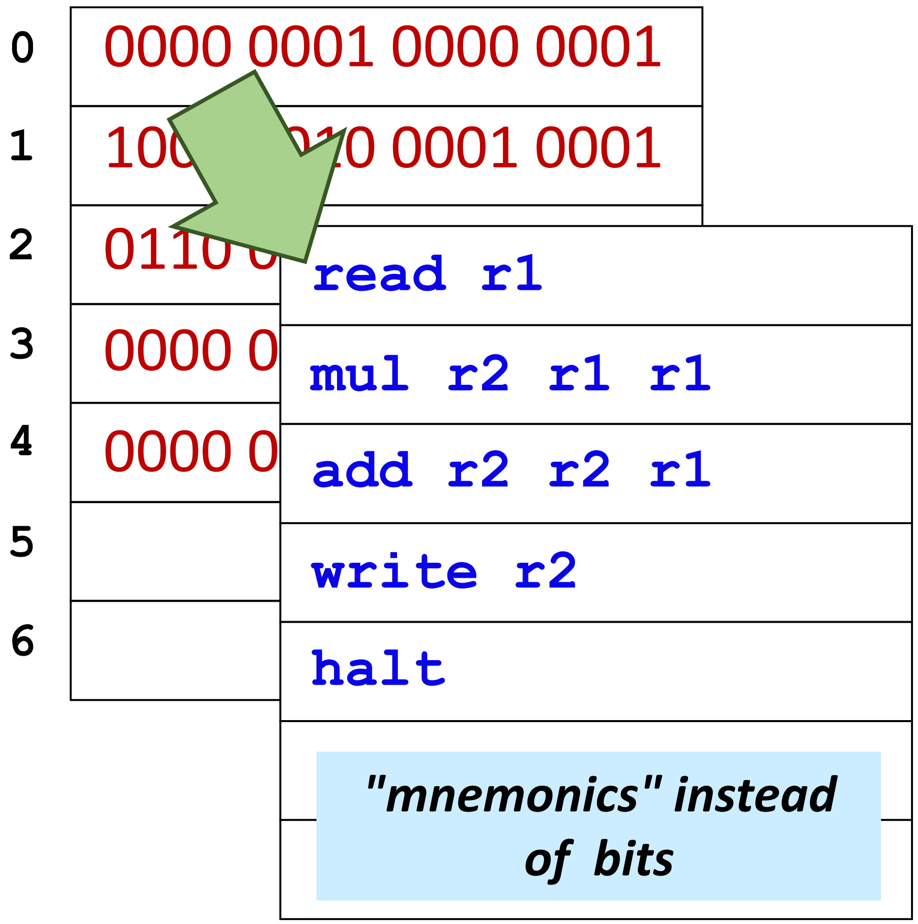
- A program written in machine language is a series of binary numbers representing the instructions stored in memory.
- The ***stored program*** concept is a key reason why computers are so powerful:
  - **Running a different program does not require large amounts of time and effort to reconfigure or rewire hardware; it only requires writing the new program to memory.**

Assembly Language

- **Assembly language** is a human-readable machine language.

- Instead of programming in binary (0's and 1's), it is easier to use an assembly language.

- An ***assembler*** is a computer program that interprets software programs written in assembly language into machine language.



The Harvey Mudd Miniature Machine (HMMM)

- Hmmm (Harvey Mudd Miniature Machine) is a 16-bit, 23-instruction simulated assembly language with 2^8=256 16-bit words of memory.

- In addition to the **program counter** and **instruction register**, there are 16 registers named `r0` through `r15`.

Hmmm assembly code

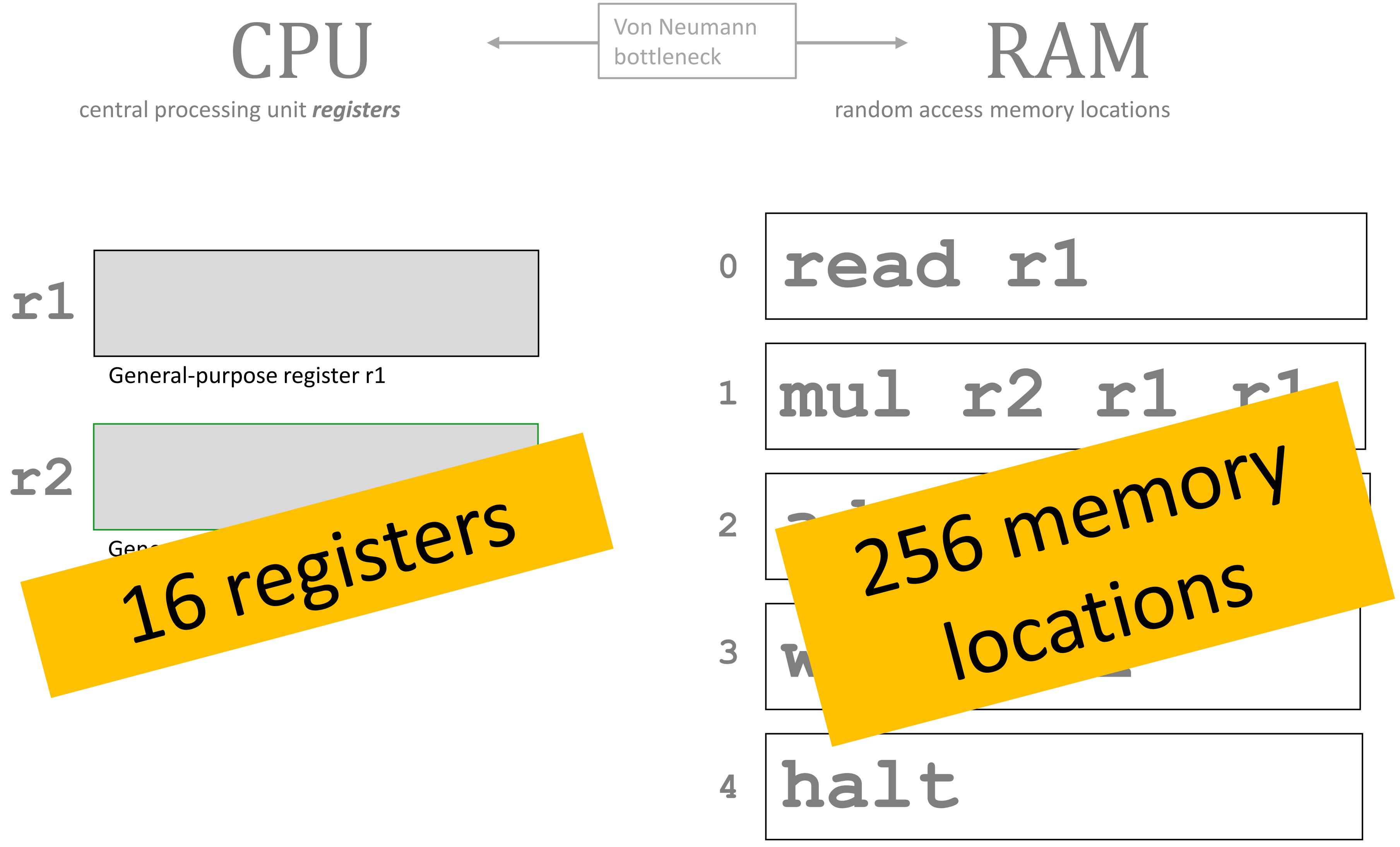
```
0    read    r1
1    read    r2
2    mul     r1 r1 r2
3    setn    r2 2
4    div     r1 r1 r2
5    write   r1
6    halt
```

→

Corresponding instructions in machine language

```
0000 0001 0000 0001
0000 0010 0000 0001
1000 0001 0001 0010
0001 0010 0000 0010
1001 0001 0001 0010
0000 0001 0000 0010
0000 0000 0000 0000
```

The Harvey Mudd Miniature Machine (HMMM)



The Harvey Mudd Miniature Machine (HMMM)

`read r1`	reads from keyboard into **`reg1`**	
`write r2`	outputs **`reg2`** onto the screen	
`setn r1 42`	`reg1 = 42`	you can replace 42 with anything from -128 to 127
`addn r1 -1`	`reg1 = reg1 - 1`	a shortcut
`add r3 r1 r2`	`reg3 = reg1 + reg2`	
`sub r3 r1 r2`	`reg3 = reg1 - reg2`	
`mul r2 r1 r1`	`reg2 = reg1 * reg1`	
`div r1 r1 r2`	`reg1 = reg1 / reg2`	integers only!

The Harvey Mudd Miniature Machine (HMMM)

Instruction	Description	Aliases
System instructions		
halt	Stop!	
read rX	Place user input in register rX	
write rX	Print contents of register rX	
nop	Do nothing	
Setting register data		
setn rX N	Set register rX equal to the integer N (-128 to +127)	
addn rX N	Add integer N (-128 to 127) to register rX	
copy rX rY	Set rX = rY	mov
Arithmetic		
add rX rY rZ	Set rX = rY + rZ	
sub rX rY rZ	Set rX = rY - rZ	
neg rX rY	Set rX = -rY	
mul rX rY rZ	Set rX = rY * rZ	
div rX rY rZ	Set rX = rY / rZ (integer division; no remainder)	
mod rX rY rZ	Set rX = rY % rZ (returns the remainder of integer division)	
Jumps!		
jumpn N	Set program counter to address N	
jumpr rX	Set program counter to address in rX	jump
jeqzn rX N	If rX == 0, then jump to line N	jeqz
jnezn rX N	If rX != 0, then jump to line N	jnez
jgtzn rX N	If rX > 0, then jump to line N	jgtz
jltzn rX N	If rX < 0, then jump to line N	jltz
calln rX N	Copy the next address into rX and then jump to mem. addr. N	call
Interacting with memory (RAM)		
loadn rX N	Load register rX with the contents of memory address N	
storen rX N	Store contents of register rX into memory address N	
loadr rX rY	Load register rX with data from the address location held in reg. rY	loadi, load
storer rX rY	Store contents of register rX into memory address held in reg. rY	storei, store

Hmmm

the complete reference

At
www.cs.hmc.edu/~cs5grad/cs5/hmmm/documentation/documentation.html

Example #1:

Screen

6 (input)

CPU

central processing unit ***registers***

Von Neumann bottleneck

RAM

random access memory locations

r1

General-purpose register r1

r2

General-purpose register r2

6

0	`read r1`
1	`mul r2 r1 r1`
2	`add r2 r2 r1`
3	`write r2`
4	`halt`

What does this program do?

Example #1 (cont.):

Screen

6 (input)

6

0	`read r1`	# Get input from user to `r1`
1	`mul r2 r1 r1`	# `r2 = r1 * r1`
2	`add r2 r2 r1`	# `r2 = r2 + r1`
3	`write r2`	# Print the contents of register `r2` on standard output
4	`halt`	# Halt program

Jumps in HMMM

`jeqzn r1 42`	IF **r1 == 0** THEN jump to line number **42**
`jgtzn r1 42`	IF **r1 > 0** THEN jump to line number **42**
`jltzn r1 42`	IF **r1 < 0** THEN jump to line number **42**
`jnezn r1 42`	IF **r1 != 0** THEN jump to line number **42**

Unconditional jump

`jumpn 42` Jump to program line # **42**

Indirect jump

`jumpr r1` Jump to the line# ***stored*** in **r1**

Example #2:

Screen

-6 (input)

RAM

0	`read r1`
1	`jgtzn r1 7`
2	`setn r2 -1`
3	`mul r1 r1 r2`
4	`nop`
5	`nop`
6	`nop`
7	`write r1`
8	`halt`

Space for future expansion!

What function does this program implement?

Exercise

1. Write a Hmmm program to compute the following for **`x`** given as user input and output the result to the screen:

   a) If **`x`**<0 `3x - 4`

   b) else if x>0 `X / 5`

   c) else X^2`+10 / 5`