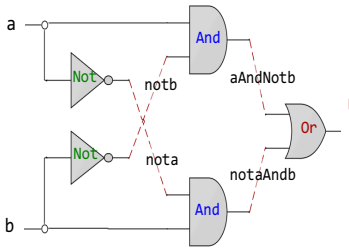
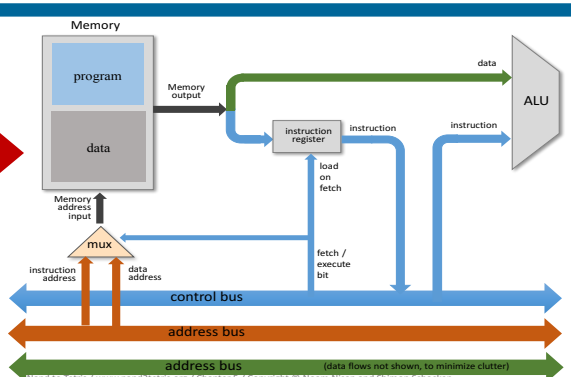
Computer Organization & Assembly Language Programming




```
CHIP Xor {
  IN a, b;
  OUT out;
  PARTS:
  Not(in=a, out=nota);
  Not(in=b, out=notb);
  And(a=nota, b=b, out=w1);
  And(a=a, b=notb, out=w2);
  Or(a=w1, b=w2, out=out);
}
```




```
@R1
D=M
@temp
M=D
```

```
0000000000000001
1111110000010000
0000000000010000
1110001100001000
```

Lecture # 19

Hack Assembly Programming - I

```
#include<stdio.h>
#include<stdlib.h>
int main(){
  printf("Learning is fun with Arif\n");
  exit(0);
}
```

```
global main
SECTION .data
   msg: db "Learning is fun with Arif", 0Ah, 0h
   len_msg: equ $ - msg
SECTION .text
   main:
      mov rax,1
      mov rdi,1
      mov rsi,msg
      mov rdx,len_msg
      syscall
      mov rax,60
      mov rdi,0
      syscall
```

```
0:  b8 01 00 00 00
5:  bf 01 00 00 00
a:  48 be 00 00 00 00 00
11: 00 00 00
14: ba 1b 00 00 00
19: 0f 05
1b: b8 3c 00 00 00
20: bf 00 00 00 00
25: 0f 05
```

Slides of first half of the course are adapted from:
https://www.nand2tetris.org
Download s/w tools required for first half of the course from the following link:
https://drive.google.com/file/d/0B9c0BdDJz6XpZUh3X2dPR1o0MUE/view

Instructor: Muhammad Arif Butt, Ph.D.

Today's Agenda

- Review of Hack Computer Assembly Instructions
- Hack Assembly Programs
- A Hello World
- CPU Emulator
- Demo
- Program Termination

Review of Hack Computer Assembly Instructions

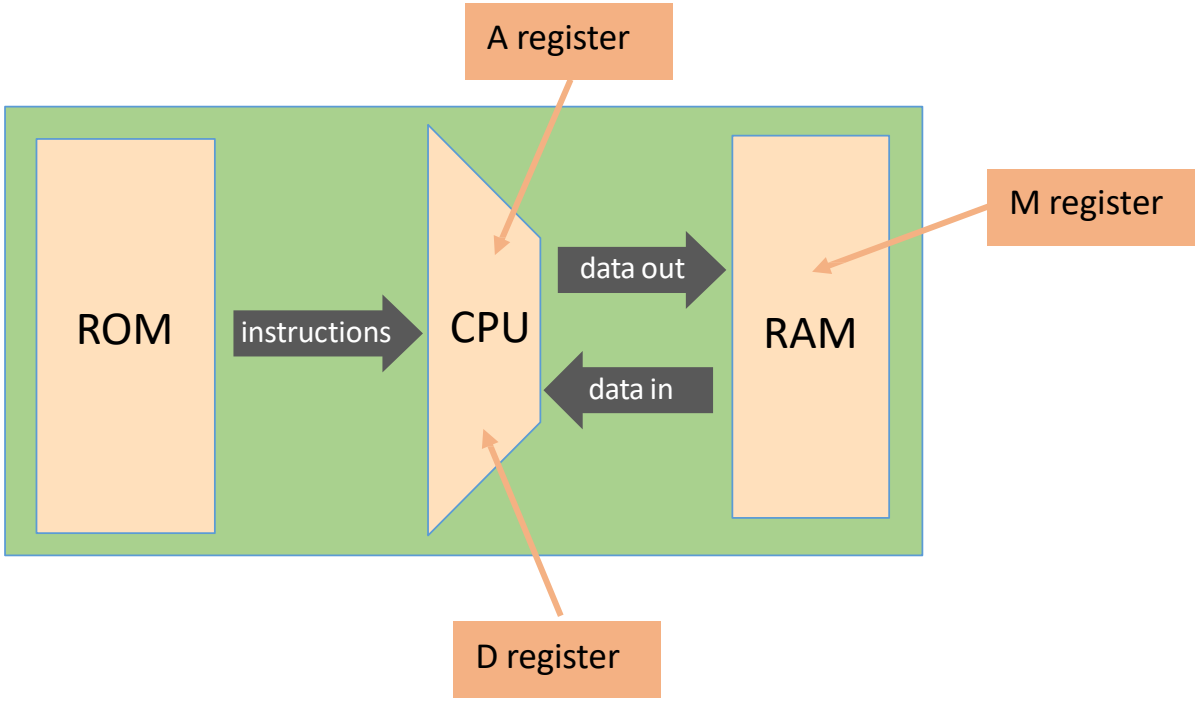
Recap: Registers and Memory

Registers:

- D: Used to hold data value
- A: Used to hold data value / address of the memory
- M: Represents the currently selected memory register, i.e., M=RAM[A]

Data memory (RAM) & Instruction memory (ROM):

- Both are a sequence of 16-bit registers having 15 bit address, i.e., 32K 16 bit words



Recap: The Hack Assembly Instructions

The A-instruction:

Syntax: `@value`

The C-instruction:

Syntax: `dest= comp ;jump`

`dest= comp`

`comp ;jump`

comp:

```
0, 1, -1, D, A, M, !D, !A, !M, -D, -A, -M,
D+1, A+1, M+1, D-1, A-1, M-1,
D+A, D-A, A-D, D&A, D|A,
D+M, D-M, M-D, D&M, D|M
```

dest: null, M, D, A, MD, AM, AD, AMD

jump: null, JGT, JEQ, JGE, JLT, JNE, JLE, JMP

```
// D=10
@10
D=A

// D++
D=D+1

// RAM[23]=65
@23
M=65

// D=RAM[17]
@17
D=M

// RAM[17]=D
@17
M=D
```

```
// RAM[17]=10
@10
D=A
@17
M=D

// RAM[5] = RAM[3]
@3
D=M
@5
M=D
```

```
// goto instr at addr 72
@72
0; JMP

// if D+1 <= 0 then jump
to instr at addr 1024
@1024
D+1 ;JLE
```

Hack Assembly Programs

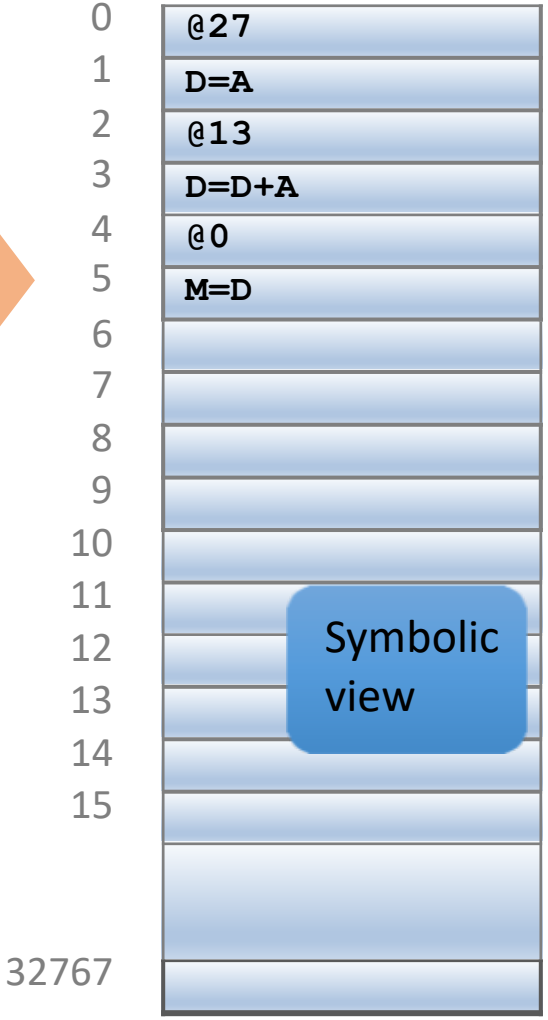
Example: addv0.asm

Hack assembly code

```
// Program: addv0.asm
// Computes: RAM[0] = 27 + 13

0   @27     // A = 27
1   D=A     // D = 27

2   @13     // A = 13

3   D=D+A   // D = 27 + 13

4   @0.     // A = 0
5   M=D     // RAM[0] = D
```

Load

Memory (ROM)



Example: addv0.asm (cont...)

Hack assembly code

```
// Program: addv0.asm
// Computes: RAM[0] = 27 + 13

0   @27     // A = 27
1   D=A     // D = 27

2   @13     // A = 13

3   D=D+A   // D = 27 + 13

4   @0.     // A = 0
5   M=D     // RAM[0] = D
```

translate and load

Memory (ROM)

0	0000000000011011
1	1110110000010000
2	0000000000001101
3	1110000010010000
4	0000000000000000
5	1110001100001000
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
32767	

Machine Code

Example: addv1.asm

Hack assembly code

```
// Program: addv1.asm
// Computes: RAM[2] = RAM[0] + RAM[1]
// Usage: put values in RAM[0], RAM[1]

0   @0
1   D=M   // D = RAM[0]

2   @1
3   D=D+M // D = D + RAM[1]

4   @2
5   M=D   // RAM[2] = D
```

translate and load

Memory (ROM)

0	0000000000000000
1	1111110000010000
2	0000000000000001
3	1111000010010000
4	0000000000000010
5	1110001100001000
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
32767	

Binary view

Executing a Hack Assembly Program

```
// Program: addv1.asm
// Computes: RAM[2] = RAM[0] + RAM[1]
// Usage: put values in RAM[0], RAM[1]

   @0
   D=M     // D = RAM[0]

   @1
   D=D+M // D = D + RAM[1]

   @2
   M=D     // RAM[2] = D
```

Hack Assembler

Translate

```
0000000000000000
1111110000010000
0000000000000001
1111000010010000
0000000000000010
1110001100001000
```

Load in Hack Computer

Execute

- We will develop Hack Assembler later in the course
- Now, we can use the CPU emulator for the purpose

Emulator

- In contrast to a simulator, an **emulator** attempt to mimic the hardware features of a production environment, as well as software features
- Emulation is the process of artificially executing code intended for a "foreign" architecture by converting it to the assembly/machine language of that CPU
- The **CPU Emulator** that we will be using is designed and developed by students of Interdisciplinary Center Herzliya Efi Arazi School of Computer Science, headed by Yaron Ukrainitz
- It is a software tool build in Java. We can load Hack assembly program into CPU emulator's instruction memory, the CPU emulator translate it into machine language and execute it
- Convenient for debugging and executing symbolic Hack programs in simulation

How to Download the CPU Emulator?

- Type the following URL in your browser:

  https://bitbucket.org/arifpucit/

- In the public repositories pane, click the ***coal-repo*** repository, containing all the source codes as well as the software tools used in this course
- In the left pane, click ***Downloads*** to download the entire repository on your system. Now on your system just check the contents of ***tools*** directory that you have just downloaded

```
Arif-MacBook:arifpucit-coal-repo/tools$ ls
HardwareSimulator.sh              HardwareSimulator.bat
CPUEmulator.sh                    CPUEmulator.bat
Assembler.sh                      Assembler.bat
VMEmulator.sh                     VMEmulator.bat
JackCompiler.sh                   JackCompiler.bat
TextComparer.sh                   TextComparer.bat
builtInChips    builtInVMCode     bin     OS
```

Starting the CPU Emulator

- Follow the following steps to start the CPU emulator on UNIX/Mac OS:
  - Open the terminal
  - Go to tools directory
  - Set execute permissions of the file `CPUEmulator.sh`
  - Execute it

```
tools — -bash — 80×24
(base) Arifs-MacBook-Pro:tools arif$ ls
Assembler.bat           JackCompiler.bat        VMEmulator.sh
Assembler.sh            JackCompiler.sh         bin
CPUEmulator.bat         OS                      builtInChips
CPUEmulator.sh          TextComparer.bat        builtInVMCode
HardwareSimulator.bat   TextComparer.sh
HardwareSimulator.sh    VMEmulator.bat
(base) Arifs-MacBook-Pro:tools arif$ chmod +x CPUEmulator.sh
(base) Arifs-MacBook-Pro:tools arif$ ./CPUEmulator.sh
```

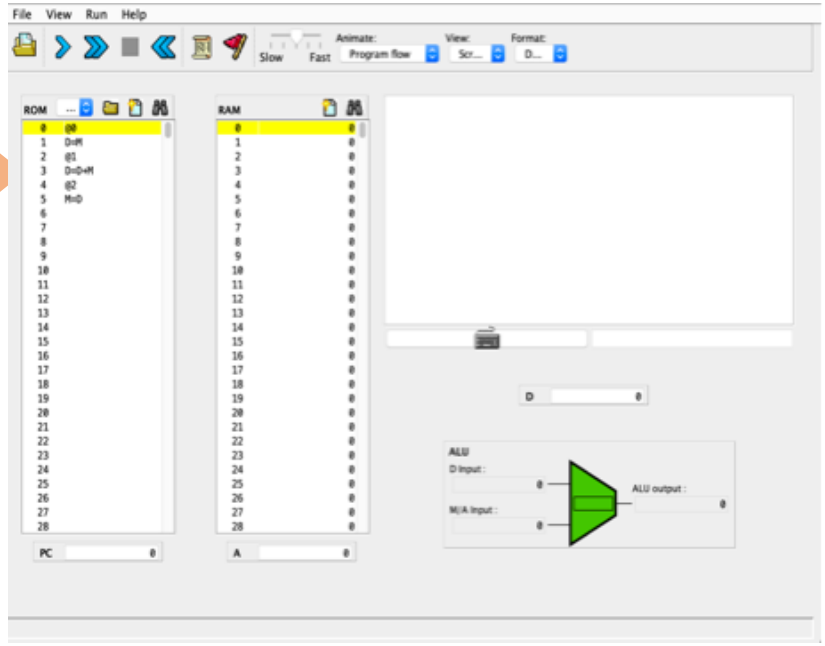
Loading an Assembly Program in CPU Emulator

Hack assembly code

```
// Program: addv1.asm
// Computes: RAM[2] = RAM[0] + RAM[1]
// Usage: put values in RAM[0], RAM[1]

0   @0
1   D=M   // D = RAM[0]

2   @1
3   D=D+M // D = D + RAM[1]

4   @2
5   M=D   // RAM[2] = D
```

Load

(the simulator software translates from symbolic to binary as it loads)



CPU Emulator

- A software tool build in Java
- We can load Hack assembly program into CPU emulator's instruction memory, the CPU emulator translate it into machine language and execute it
- Convenient for debugging and executing symbolic Hack programs in simulation

Running an Assembly Program in CPU Emulator

Program Termination

Terminating a Program

Hack assembly code

```
// Program: addv1.asm
// Computes: RAM[2] = RAM[0] + RAM[1]
// Usage: put values in RAM[0], RAM[1]

0   @0
1   D=M   // D = RAM[0]

2   @1
3   D=D+M // D = D + RAM[1]

4   @2
5   M=D   // RAM[2] = D
```

Memory (ROM)

If you load above program inside the CPU emulator and run it using fast forward button. The computer continues to execute the program from instruction at address 0-5 and then continues executing onwards and does not halt

Terminating a Program

Hack assembly code

```
// Program: addv2.asm
// Computes: RAM[2] = RAM[0] + RAM[1]
// Usage: put values in RAM[0], RAM[1]

0   @0
1   D=M   // D = RAM[0]

2   @1
3   D=D+M // D = D + RAM[1]
4   @2
5   M=D   // RAM[2] = D


6   @6
7   0;JMP
```

- Jump to instruction number `A` (which happens to be 6)
- `0`: syntax convention for `jmp` instructions

Memory (ROM)

0	`@0`
1	`D=M`
2	`@1`
3	`D=D+M`
4	`@2`
5	`M=D`
6	`@6`
7	`0;JMP`
8	
9	
10	
11	
12	
13	
14	
15	

Best practice:

Remember computers never stand still. They always need to do some thing, i.e., execute some instruction.

To terminate a program safely, end it with an infinite loop.

Running an Assembly Program in CPU Emulator

Things To Do

- Download CPU emulator along with other tools and programs from https://bitbucket.org/arifpucit/ and run it on your system (Mac, Linux, Windows)
- Download all the assembly programs from the course bitbucket repository, make changes to them and execute them in the CPU Emulator
- Must build a very clear understanding of the concepts discussed in today's session
- Interested students should try to write down mulv0.asm program similar to addv0.asm

Coming to office hours does NOT mean you are academically week!