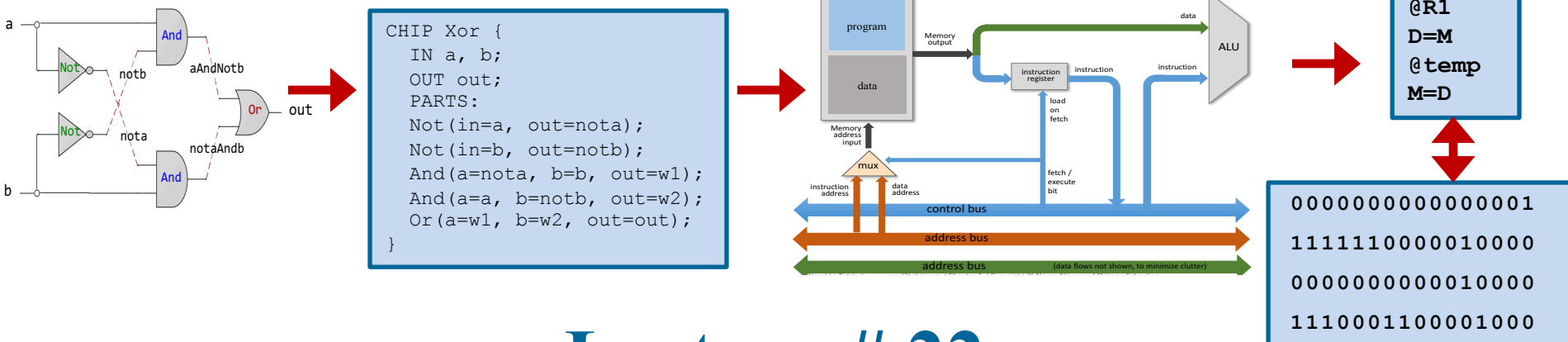




Digital Logic Design



Lecture # 22

Hack Machine Language

```
#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.





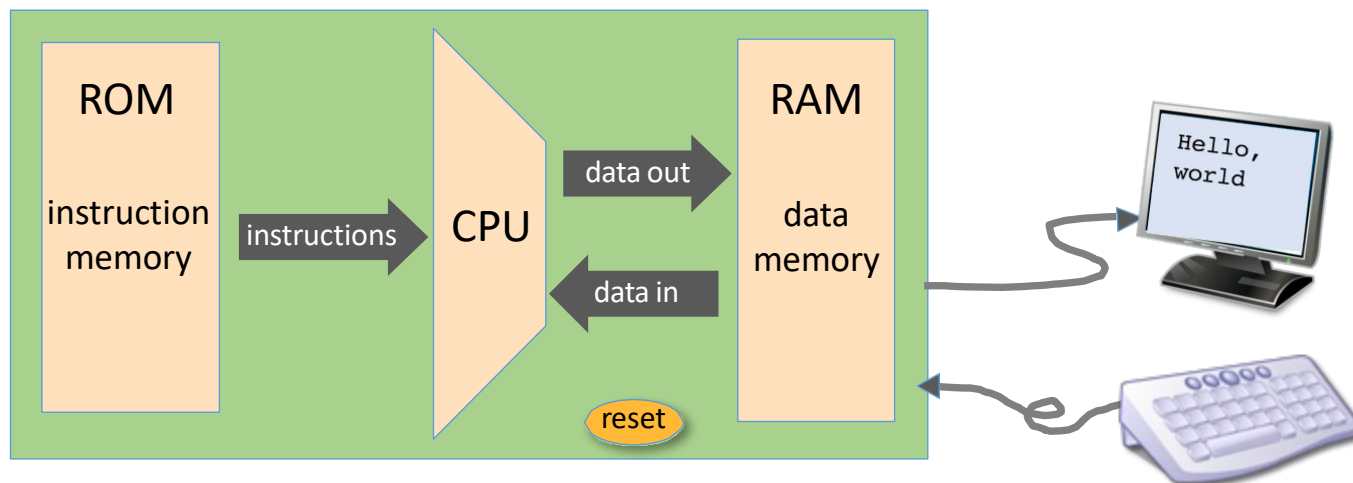
Today's Agenda

- Hack Computer Machine Language
- Review of h/w of Hack Computer
- Software of Hack Computer
 - A Instruction
 - C Instruction
 - Examples
- Binary Code Format of Hack Computer Instruction
 - Encoding of 16 bit A-Instruction
 - Encoding of 16 bit C-Instruction
 - Examples
- A Complete Hack Program: Assembly Language & Machine Code





Hack Computer: Hardware

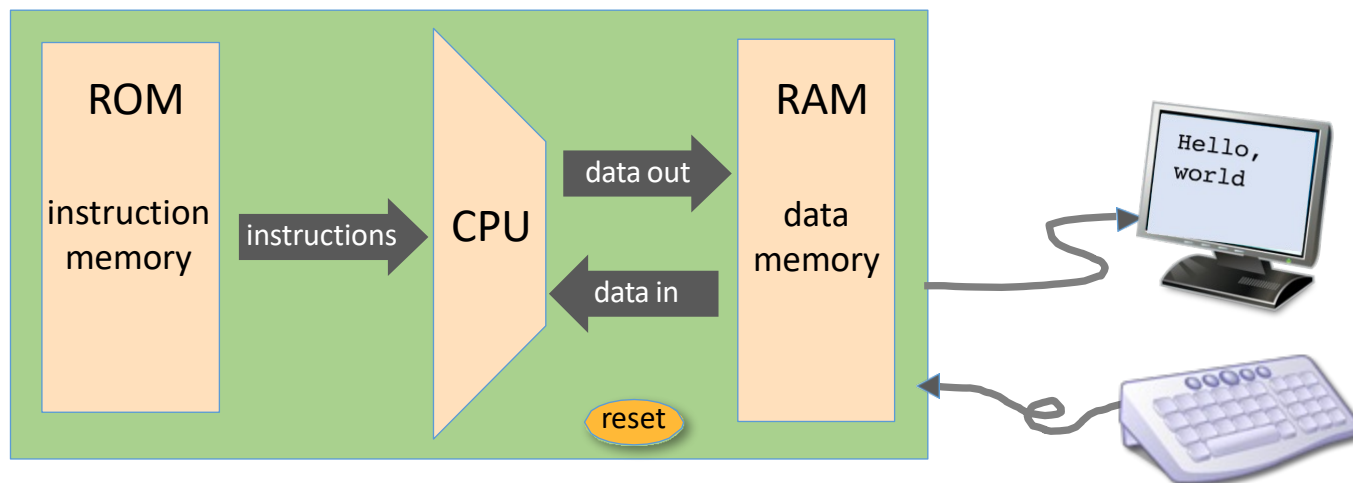


Hack computer is a 16-bit machine consisting of:

- Central Processing Unit (CPU): performs 16-bit instructions
- **Data memory (RAM): a sequence of 16-bit registers having 15 bit addr:**
$$\text{RAM}[0], \text{RAM}[1], \text{RAM}[2], \dots$$
- **Instruction memory (ROM): a sequence of 16-bit registers having 15 bit addr:**
$$\text{ROM}[0], \text{ROM}[1], \text{ROM}[2], \dots$$
- Two memory-mapped I/O devices: a screen and a keyboard
- Instruction bus / data bus / address buses



Hack Computer: Software



Hack machine language:

- 16-bit A-instructions
- 16-bit C-instructions

Hack program:

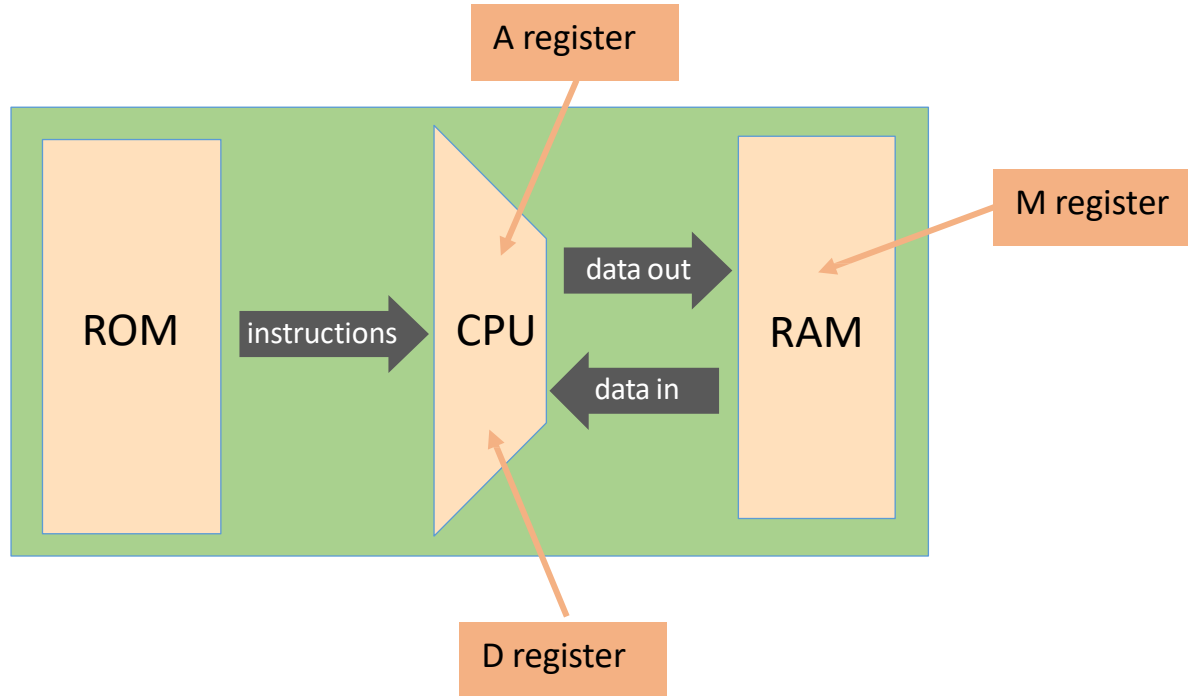
- A sequence of instructions written in the hack machine language

Control:

- The ROM is loaded with a Hack program (16 bit instructions)
- The reset button is pushed
- The program starts running



Hack Computer: Registers



The Hack machine language recognizes three 16-bit registers:

- D: used to hold data value
- A: used to hold data value / address of the memory
- M: represents the currently selected memory register: $M = \text{RAM}[A]$



The A-Instruction

The A-instruction is used to set the A register to a 15 bit value

Syntax: `@ value` Where value is either:

- A non-negative decimal constant ($\leq 2^{15} - 1$) or
- A symbol referring to such a constant

Semantics: Sets the A register to value, so after this

- $\text{RAM}[A]$ becomes the selected RAM register
- $\text{ROM}[A]$ becomes the selected ROM register

Example: `@17 //A =17`

- Sets the register A to the value of 17
- As a side effect the $\text{RAM}[17]$ becomes the selected RAM register



The C Instruction

Syntax: `dest= comp ; jump` (either dest or jump fields may be empty)

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

Working:

A C instruction can be used in either of the following two ways:

- Store the result at some destination
- Use the result of the computation to jump

`dest= comp`

`comp ; jump`



Examples: Hack Machine 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

Example 1: Set register D to a value of -1

D=-1

dest= comp



Examples: Hack Machine 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

Example 2: Suppose the programmer wants to increment the value of D

`D=D+1`

`dest= comp`



Examples: Hack Machine 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

Example 3: Suppose the programmer wants to add the contents of D and A-register and place the result in D-register

`D=D+A`

`dest= comp`



Examples: Hack Machine 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

Example 4: Suppose the programmer wants to store a number 10 in register D

`@10 //A =10`

`D=A`

`@value`

`dest= comp`



Examples: Hack Machine 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

Example 5: Suppose the programmer wants to write the value of register D at RAM[135]

`@135 //A =135`

`M=D`

`@value`

`dest= comp`



Examples: Hack Machine 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

Example 6: Suppose the programmer wants to write the value of register D+1 at RAM[135]

`@135 //A =135`

`M=D+1`

`@value`

`dest= comp`



Examples: Hack Machine 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

Example 7: Suppose the programmer wants to read memory contents from address 325 and place them in D register

`@325 //A =325`
`D=M //D=M[325]`

`@value`
`dest= comp`



Examples: Hack Machine 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**

Example 8: Suppose the programmer wants to do an unconditional jump to ROM[431]

`@431 //A =431`

`0 ; JMP`

`@value`

`comp ; jump`



Examples: Hack Machine 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

Example 9: Suppose the programmer wants to jump to ROM[97], if $D-1 == 0$

`@97 //A =97`

`D-1 ; JEQ`

`@value`

`comp ; jump`



Examples: Hack Machine 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

Example 10: Suppose the programmer wants to write constant 54 at RAM[17]

```
//D=54
@54
D=A
//M[17]=D
@17
M=D
```

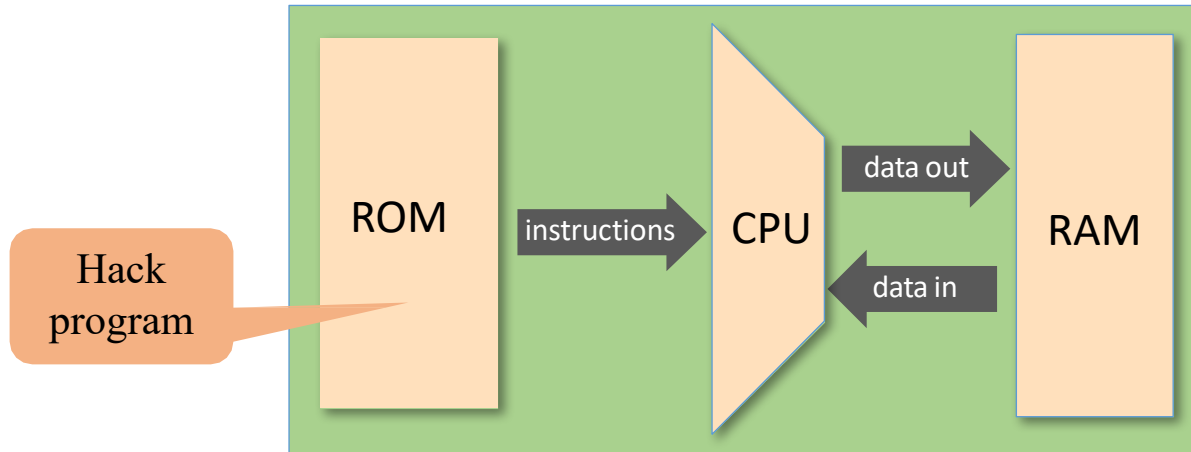


Hack

Machine Language vs. Symbolic Language



Hack Machine Language



Two ways to express the same semantics:

Symbolic:

```
@17  
D+1;JLE
```

translate

Binary:

```
00000000000010001  
1110011111000110
```

load & execute



Binary Code Format: A-Instruction

- The A-instruction is used to set the A register to a 15 bit value
- The symbolic as well as machine code for the instruction is shown below:

Symbolic Code

@23

Machine Code

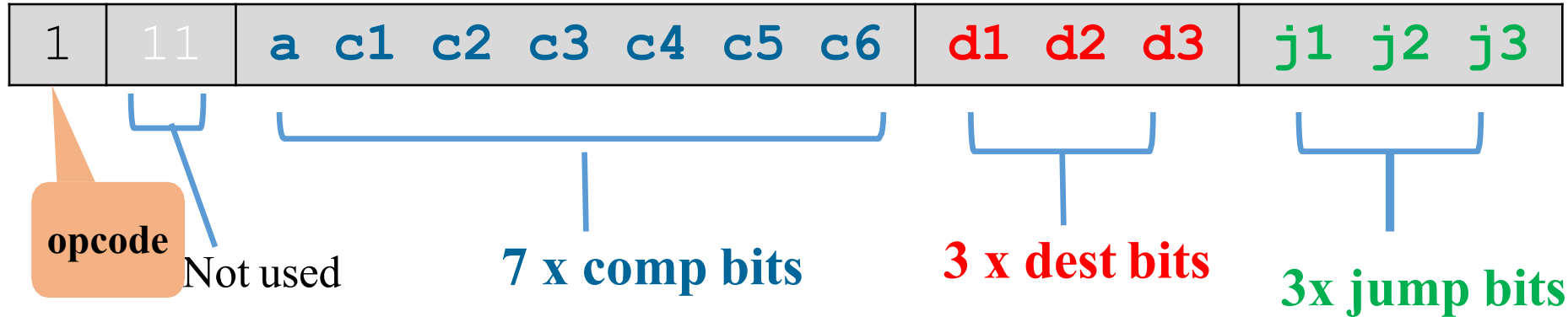
0000 0000 0001 0111

opcode signifying
an A-instruction



Binary Code Format: C-Instruction

dest= comp ; jump





Binary Code Format: C-Instruction

dest= comp ; jump

1	11	a c1 c2 c3 c4 c5 c6	d1 d2 d3	j1 j2 j3
---	----	---------------------	----------	----------

<i>comp</i>		c1	c2	c3	c4	c5	c6
0		1	0	1	0	1	0
1		1	1	1	1	1	1
-1		1	1	1	0	1	0
D		0	0	1	1	0	0
A	M	1	1	0	0	0	0
!D		0	0	1	1	0	1
!A	!M	1	1	0	0	0	1
-D		0	0	1	1	1	1
-A	-M	1	1	0	0	1	1
D+1		0	1	1	1	1	1
A+1	M+1	1	1	0	1	1	1
D-1		0	0	1	1	1	0
A-1	M-1	1	1	0	0	1	0
D+A	D+M	0	0	0	0	1	0
D-A	D-M	0	1	0	0	1	1
A-D	M-D	0	0	0	1	1	1
D&A	D&M	0	0	0	0	0	0
D A	D M	0	1	0	1	0	1
a==0	a==1						



Binary Code Format: C-Instruction

dest= comp ; jump

1	11	a c1 c2 c3 c4 c5 c6	d1 d2 d3	j1 j2 j3
---	----	---------------------	----------	----------

<i>dest</i>	d1 d2 d3	effect: the value is stored in:
null	0 0 0	The value is not stored
M	0 0 1	RAM[A]
D	0 1 0	D register
MD	0 1 1	RAM[A] and D register
A	1 0 0	A register
AM	1 0 1	A register and RAM[A]
AD	1 1 0	A register and D register
AMD	1 1 1	A register, RAM[A], and D register



Binary Code Format: C-Instruction

dest= comp ; jump

1	11	a c1 c2 c3 c4 c5 c6	d1 d2 d3	j1 j2 j3
---	----	---------------------	----------	----------

<i>jump</i>	j1 j2 j3	effect
null	0 0 0	no jump
JGT	0 0 1	if out>0 jump
JEQ	0 1 0	if out=0 jump
JGE	0 1 1	if out≥0 jump
JLT	1 0 0	if out<0 jump
JNE	1 0 1	if out≠0 jump
JLE	1 1 0	if out≤0 jump
JMP	1 1 1	unconditional jump



Complete Specification of C-Instruction

dest= comp ; jump

1	11	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
---	----	---	----	----	----	----	----	----	----	----	----	----	----	----

<i>comp</i>		c1	c2	c3	c4	c5	c6
0		1	0	1	0	1	0
1		1	1	1	1	1	1
-1		1	1	1	0	1	0
D		0	0	1	1	0	0
A	M	1	1	0	0	0	0
!D		0	0	1	1	0	1
!A	!M	1	1	0	0	0	1
-D		0	0	1	1	1	1
-A	-M	1	1	0	0	1	1
D+1		0	1	1	1	1	1
A+1	M+1	1	1	0	1	1	1
D-1		0	0	1	1	1	0
A-1	M-1	1	1	0	0	1	0
D+A	D+M	0	0	0	0	1	0
D-A	D-M	0	1	0	0	1	1
A-D	M-D	0	0	0	1	1	1
D&A	D&M	0	0	0	0	0	0
D A	D M	0	1	0	1	0	1
a==0	a==1						

<i>dest</i>	d1	d2	d3	effect: the value is stored in:
null	0	0	0	The value is not stored
M	0	0	1	RAM[A]
D	0	1	0	D register
MD	0	1	1	RAM[A] and D register
A	1	0	0	A register
AM	1	0	1	A register and RAM[A]
AD	1	1	0	A register and D register
AMD	1	1	1	A register, RAM[A], and D register

<i>jump</i>	j1	j2	j3	effect:
null	0	0	0	no jump
JGT	0	0	1	if out > 0 jump
JEQ	0	1	0	if out = 0 jump
JGE	0	1	1	if out ≥ 0 jump
JLT	1	0	0	if out < 0 jump
JNE	1	0	1	if out ≠ 0 jump
JLE	1	1	0	if out ≤ 0 jump
JMP	1	1	1	Unconditional jump



Example 1: Symbolic Code to Binary Code

dest= comp ; jump

1	11	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
---	----	---	----	----	----	----	----	----	----	----	----	----	----	----

<i>comp</i>		c1	c2	c3	c4	c5	c6
0		1	0	1	0	1	0
1		1	1	1	1	1	1
-1		1	1	1	0	1	0
D		0	0	1	1	0	0
A	M	1	1	0	0	0	0
!D		0	0	1	1	0	1
!A	!M	1	1	0	0	0	1
-D		0	0	1	1	1	1
-A	-M	1	1	0	0	1	1
D+1		0	1	1	1	1	1
A+1	M+1	1	1	0	1	1	1
D-1		0	0	1	1	1	0
A-1	M-1	1	1	0	0	1	0
D+A	D+M	0	0	0	0	1	0
D-A	D-M	0	1	0	0	1	1
A-D	M-D	0	0	0	1	1	1
D&A	D&M	0	0	0	0	0	0
D A	D M	0	1	0	1	0	1
a==0	a==1						

<i>dest</i>	d1	d2	d3	effect: the value is stored in:
null	0	0	0	The value is not stored
M	0	0	1	RAM[A]
D	0	1	0	D register
MD	0	1	1	RAM[A] and D register
A	1	0	0	A register
AM	1	0	1	A register and RAM[A]
AD	1	1	0	A register and D register
AMD	1	1	1	A register, RAM[A], and D register

<i>jump</i>	j1	j2	j3	effect:
null	0	0	0	no jump
JGT	0	0	1	if out > 0 jump
JEQ	0	1	0	if out = 0 jump
JGE	0	1	1	if out ≥ 0 jump
JLT	1	0	0	if out < 0 jump
JNE	1	0	1	if out ≠ 0 jump
JLE	1	1	0	if out ≤ 0 jump
JMP	1	1	1	Unconditional jump

Example 1: Suppose the programmer wants to increment the value of D

D= D+1

1 11 0011111 010 000



Example 2: Symbolic Code to Binary Code

dest= comp ; jump

1	11	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
---	----	---	----	----	----	----	----	----	----	----	----	----	----	----

<i>comp</i>		c1	c2	c3	c4	c5	c6
0		1	0	1	0	1	0
1		1	1	1	1	1	1
-1		1	1	1	0	1	0
D		0	0	1	1	0	0
A	M	1	1	0	0	0	0
!D		0	0	1	1	0	1
!A	!M	1	1	0	0	0	1
-D		0	0	1	1	1	1
-A	-M	1	1	0	0	1	1
D+1		0	1	1	1	1	1
A+1	M+1	1	1	0	1	1	1
D-1		0	0	1	1	1	0
A-1	M-1	1	1	0	0	1	0
D+A	D+M	0	0	0	0	1	0
D-A	D-M	0	1	0	0	1	1
A-D	M-D	0	0	0	1	1	1
D&A	D&M	0	0	0	0	0	0
D A	D M	0	1	0	1	0	1
a==0	a==1						

<i>dest</i>	d1	d2	d3	effect: the value is stored in:
null	0	0	0	The value is not stored
M	0	0	1	RAM[A]
D	0	1	0	D register
MD	0	1	1	RAM[A] and D register
A	1	0	0	A register
AM	1	0	1	A register and RAM[A]
AD	1	1	0	A register and D register
AMD	1	1	1	A register, RAM[A], and D register

<i>jump</i>	j1	j2	j3	effect:
null	0	0	0	no jump
JGT	0	0	1	if out > 0 jump
JEQ	0	1	0	if out = 0 jump
JGE	0	1	1	if out ≥ 0 jump
JLT	1	0	0	if out < 0 jump
JNE	1	0	1	if out ≠ 0 jump
JLE	1	1	0	if out ≤ 0 jump
JMP	1	1	1	Unconditional jump

Example 2: Suppose the programmer wants to add the contents of D and A-register and place the result in D-register

D=D+A

1 11 0000010 010 000



Example 3: Symbolic Code to Binary Code

dest= comp ; jump

1	11	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
---	----	---	----	----	----	----	----	----	----	----	----	----	----	----

<i>comp</i>		c1	c2	c3	c4	c5	c6
0		1	0	1	0	1	0
1		1	1	1	1	1	1
-1		1	1	1	0	1	0
D		0	0	1	1	0	0
A	M	1	1	0	0	0	0
!D		0	0	1	1	0	1
!A	!M	1	1	0	0	0	1
-D		0	0	1	1	1	1
-A	-M	1	1	0	0	1	1
D+1		0	1	1	1	1	1
A+1	M+1	1	1	0	1	1	1
D-1		0	0	1	1	1	0
A-1	M-1	1	1	0	0	1	0
D+A	D+M	0	0	0	0	1	0
D-A	D-M	0	1	0	0	1	1
A-D	M-D	0	0	0	1	1	1
D&A	D&M	0	0	0	0	0	0
D A	D M	0	1	0	1	0	1
a==0	a==1						

<i>dest</i>	d1	d2	d3	effect: the value is stored in:
null	0	0	0	The value is not stored
M	0	0	1	RAM[A]
D	0	1	0	D register
MD	0	1	1	RAM[A] and D register
A	1	0	0	A register
AM	1	0	1	A register and RAM[A]
AD	1	1	0	A register and D register
AMD	1	1	1	A register, RAM[A], and D register

<i>jump</i>	j1	j2	j3	effect:
null	0	0	0	no jump
JGT	0	0	1	if out > 0 jump
JEQ	0	1	0	if out = 0 jump
JGE	0	1	1	if out ≥ 0 jump
JLT	1	0	0	if out < 0 jump
JNE	1	0	1	if out ≠ 0 jump
JLE	1	1	0	if out ≤ 0 jump
JMP	1	1	1	Unconditional jump

Example 3: Suppose the programmer wants to store a number 10 in register D

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

```
0 0000000000001010
1 11 0110000 010 000
```



Example 4: Symbolic Code to Binary Code

dest= comp ; jump

1	11	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
---	----	---	----	----	----	----	----	----	----	----	----	----	----	----

<i>comp</i>		c1	c2	c3	c4	c5	c6
0		1	0	1	0	1	0
1		1	1	1	1	1	1
-1		1	1	1	0	1	0
D		0	0	1	1	0	0
A	M	1	1	0	0	0	0
!D		0	0	1	1	0	1
!A	!M	1	1	0	0	0	1
-D		0	0	1	1	1	1
-A	-M	1	1	0	0	1	1
D+1		0	1	1	1	1	1
A+1	M+1	1	1	0	1	1	1
D-1		0	0	1	1	1	0
A-1	M-1	1	1	0	0	1	0
D+A	D+M	0	0	0	0	1	0
D-A	D-M	0	1	0	0	1	1
A-D	M-D	0	0	0	1	1	1
D&A	D&M	0	0	0	0	0	0
D A	D M	0	1	0	1	0	1
a==0	a==1						

<i>dest</i>	d1	d2	d3	effect: the value is stored in:
null	0	0	0	The value is not stored
M	0	0	1	RAM[A]
D	0	1	0	D register
MD	0	1	1	RAM[A] and D register
A	1	0	0	A register
AM	1	0	1	A register and RAM[A]
AD	1	1	0	A register and D register
AMD	1	1	1	A register, RAM[A], and D register

<i>jump</i>	j1	j2	j3	effect:
null	0	0	0	no jump
JGT	0	0	1	if out > 0 jump
JEQ	0	1	0	if out = 0 jump
JGE	0	1	1	if out ≥ 0 jump
JLT	1	0	0	if out < 0 jump
JNE	1	0	1	if out ≠ 0 jump
JLE	1	1	0	if out ≤ 0 jump
JMP	1	1	1	Unconditional jump

Example 4: Suppose the programmer wants to write the value of register D at RAM[135]

@135 //A =135

M=D

0 000000010000111

1 11 0001100 001 000



Example 5: Symbolic Code to Binary Code

dest= comp ; jump

1	11	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
---	----	---	----	----	----	----	----	----	----	----	----	----	----	----

<i>comp</i>		c1	c2	c3	c4	c5	c6
0		1	0	1	0	1	0
1		1	1	1	1	1	1
-1		1	1	1	0	1	0
D		0	0	1	1	0	0
A	M	1	1	0	0	0	0
!D		0	0	1	1	0	1
!A	!M	1	1	0	0	0	1
-D		0	0	1	1	1	1
-A	-M	1	1	0	0	1	1
D+1		0	1	1	1	1	1
A+1	M+1	1	1	0	1	1	1
D-1		0	0	1	1	1	0
A-1	M-1	1	1	0	0	1	0
D+A	D+M	0	0	0	0	1	0
D-A	D-M	0	1	0	0	1	1
A-D	M-D	0	0	0	1	1	1
D&A	D&M	0	0	0	0	0	0
D A	D M	0	1	0	1	0	1
a==0	a==1						

<i>dest</i>	d1	d2	d3	effect: the value is stored in:
null	0	0	0	The value is not stored
M	0	0	1	RAM[A]
D	0	1	0	D register
MD	0	1	1	RAM[A] and D register
A	1	0	0	A register
AM	1	0	1	A register and RAM[A]
AD	1	1	0	A register and D register
AMD	1	1	1	A register, RAM[A], and D register

<i>jump</i>	j1	j2	j3	effect:
null	0	0	0	no jump
JGT	0	0	1	if out > 0 jump
JEQ	0	1	0	if out = 0 jump
JGE	0	1	1	if out ≥ 0 jump
JLT	1	0	0	if out < 0 jump
JNE	1	0	1	if out ≠ 0 jump
JLE	1	1	0	if out ≤ 0 jump
JMP	1	1	1	Unconditional jump

Example 5: Set RAM[64] to the value of D-register +1

@ 64
M= D+1

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
1 11 0011111 001 000



Example 6: Symbolic Code to Binary Code

dest= comp ; jump

1	11	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
---	----	---	----	----	----	----	----	----	----	----	----	----	----	----

<i>comp</i>		c1	c2	c3	c4	c5	c6
0		1	0	1	0	1	0
1		1	1	1	1	1	1
-1		1	1	1	0	1	0
D		0	0	1	1	0	0
A	M	1	1	0	0	0	0
!D		0	0	1	1	0	1
!A	!M	1	1	0	0	0	1
-D		0	0	1	1	1	1
-A	-M	1	1	0	0	1	1
D+1		0	1	1	1	1	1
A+1	M+1	1	1	0	1	1	1
D-1		0	0	1	1	1	0
A-1	M-1	1	1	0	0	1	0
D+A	D+M	0	0	0	0	1	0
D-A	D-M	0	1	0	0	1	1
A-D	M-D	0	0	0	1	1	1
D&A	D&M	0	0	0	0	0	0
D A	D M	0	1	0	1	0	1
a==0	a==1						

<i>dest</i>	d1	d2	d3	effect: the value is stored in:
null	0	0	0	The value is not stored
M	0	0	1	RAM[A]
D	0	1	0	D register
MD	0	1	1	RAM[A] and D register
A	1	0	0	A register
AM	1	0	1	A register and RAM[A]
AD	1	1	0	A register and D register
AMD	1	1	1	A register, RAM[A], and D register

<i>jump</i>	j1	j2	j3	effect:
null	0	0	0	no jump
JGT	0	0	1	if out > 0 jump
JEQ	0	1	0	if out = 0 jump
JGE	0	1	1	if out ≥ 0 jump
JLT	1	0	0	if out < 0 jump
JNE	1	0	1	if out ≠ 0 jump
JLE	1	1	0	if out ≤ 0 jump
JMP	1	1	1	Unconditional jump

Example 6: Suppose the programmer wants to read memory contents from address 25 and place them in D register

@25

D=M

0000000000011001

1 11 1110000 010 000



Example 7: Symbolic Code to Binary Code

dest= comp ; jump

1	11	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
---	----	---	----	----	----	----	----	----	----	----	----	----	----	----

<i>comp</i>		c1	c2	c3	c4	c5	c6
0		1	0	1	0	1	0
1		1	1	1	1	1	1
-1		1	1	1	0	1	0
D		0	0	1	1	0	0
A	M	1	1	0	0	0	0
!D		0	0	1	1	0	1
!A	!M	1	1	0	0	0	1
-D		0	0	1	1	1	1
-A	-M	1	1	0	0	1	1
D+1		0	1	1	1	1	1
A+1	M+1	1	1	0	1	1	1
D-1		0	0	1	1	1	0
A-1	M-1	1	1	0	0	1	0
D+A	D+M	0	0	0	0	1	0
D-A	D-M	0	1	0	0	1	1
A-D	M-D	0	0	0	1	1	1
D&A	D&M	0	0	0	0	0	0
D A	D M	0	1	0	1	0	1
a==0	a==1						

<i>dest</i>	d1	d2	d3	effect: the value is stored in:
null	0	0	0	The value is not stored
M	0	0	1	RAM[A]
D	0	1	0	D register
MD	0	1	1	RAM[A] and D register
A	1	0	0	A register
AM	1	0	1	A register and RAM[A]
AD	1	1	0	A register and D register
AMD	1	1	1	A register, RAM[A], and D register

<i>jump</i>	j1	j2	j3	effect:
null	0	0	0	no jump
JGT	0	0	1	if out > 0 jump
JEQ	0	1	0	if out = 0 jump
JGE	0	1	1	if out ≥ 0 jump
JLT	1	0	0	if out < 0 jump
JNE	1	0	1	if out ≠ 0 jump
JLE	1	1	0	if out ≤ 0 jump
JMP	1	1	1	Unconditional jump

Example 7: Suppose the programmer wants to do an unconditional jump to ROM[31]

```
@31 //A =31
0 ; JMP
```

```
0000000000011111
1 11 0101010 000 111
```




Example 8: Symbolic Code to Binary Code

dest= comp ; jump

1	11	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
---	----	---	----	----	----	----	----	----	----	----	----	----	----	----

<i>comp</i>		c1	c2	c3	c4	c5	c6
0		1	0	1	0	1	0
1		1	1	1	1	1	1
-1		1	1	1	0	1	0
D		0	0	1	1	0	0
A	M	1	1	0	0	0	0
!D		0	0	1	1	0	1
!A	!M	1	1	0	0	0	1
-D		0	0	1	1	1	1
-A	-M	1	1	0	0	1	1
D+1		0	1	1	1	1	1
A+1	M+1	1	1	0	1	1	1
D-1		0	0	1	1	1	0
A-1	M-1	1	1	0	0	1	0
D+A	D+M	0	0	0	0	1	0
D-A	D-M	0	1	0	0	1	1
A-D	M-D	0	0	0	1	1	1
D&A	D&M	0	0	0	0	0	0
D A	D M	0	1	0	1	0	1
a==0	a==1						

<i>dest</i>	d1	d2	d3	effect: the value is stored in:
null	0	0	0	The value is not stored
M	0	0	1	RAM[A]
D	0	1	0	D register
MD	0	1	1	RAM[A] and D register
A	1	0	0	A register
AM	1	0	1	A register and RAM[A]
AD	1	1	0	A register and D register
AMD	1	1	1	A register, RAM[A], and D register

<i>jump</i>	j1	j2	j3	effect:
null	0	0	0	no jump
JGT	0	0	1	if out > 0 jump
JEQ	0	1	0	if out = 0 jump
JGE	0	1	1	if out ≥ 0 jump
JLT	1	0	0	if out < 0 jump
JNE	1	0	1	if out ≠ 0 jump
JLE	1	1	0	if out ≤ 0 jump
JMP	1	1	1	Unconditional jump

Example 8: Suppose the programmer wants to jump to ROM[17], if D-1 == 0

@17 //A =17
D-1; JEQ

0 000000000010001
1 11 0001110 000 010



A Complete Hack Program

```
// Computes RAM[1] = 1+...+RAM[0]
// Usage: put a number in RAM[0]
@16
M=1      // RAM[16] = 1 (i)

@17
M=0      // RAM[17] = 0 (sum)

@16
D=M
@0
D=D-M
@17      // if i>RAM[0] goto 17
D;JGT

@16
D=M
@17
M=D+M    // sum += i
@16
M=M+1    // i++
@4       // goto 4 loop
0;JMP

@17
D=M
@1
M=D      //RAM[1] = sum
@21      //program end
0;JMP    // infinite loop
```

Observations:

- Hack program: A sequence of Hack instructions
- White space is permitted
- Comments are welcome
- There are better ways to write symbolic Hack programs
- More to come....

No need to understand ...
we'll review the code in
later part of the course



Hack Program: Assembly & Machine Code

```
// Computes RAM[1] = 1+...+RAM[0]
// Usage: put a number in RAM[0]
@16
M=1      // RAM[16] = 1 (i)

@17
M=0      // RAM[17] = 0 (sum)

@16
D=M
@0
D=D-M
@17      // if i>RAM[0] goto 17
D;JGT

@16
D=M
@17
M=D+M    // sum += i
@16
M=M+1    // i++
@4       // goto 4 loop
0;JMP

@17
D=M
@1
M=D      //RAM[1] = sum
@21      //program end
0;JMP    // infinite loop
```

translate

```
00000000000010000
11101111111001000
00000000000010001
11101010100010000
00000000000010000
11111100000010000
00000000000000000
11110100110100000
00000000000010001
11100011000000001
00000000000010000
11111100000010000
00000000000010001
11110000100010000
00000000000010000
11111101110010000
00000000000000100
11101010100001111
00000000000010001
11111100000010000
00000000000000001
11100011000010000
00000000000010101
11101010100001111
```

execute

24 Assembly Instructions mapped to 24 Machine Instructions



Things To Do

