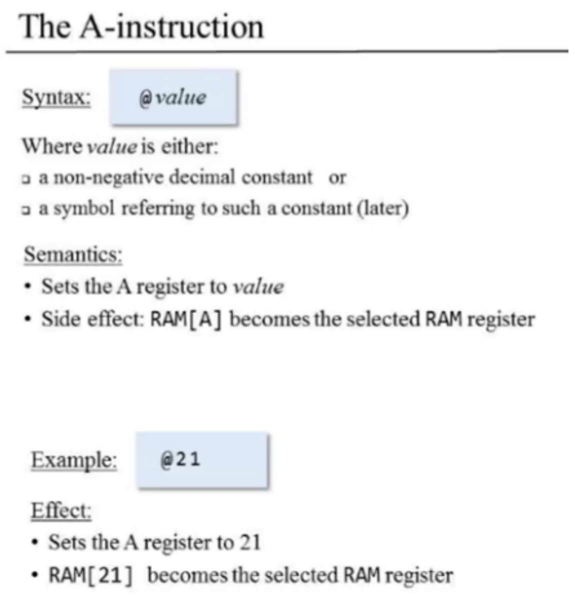
Lab #5 - Machine Language Basics

Name: Alexandra Burke
Section/Time: CS 220 1319
Date: 7/7/2023

Recall the two Assembly Instructions, A and C:



The A-instruction

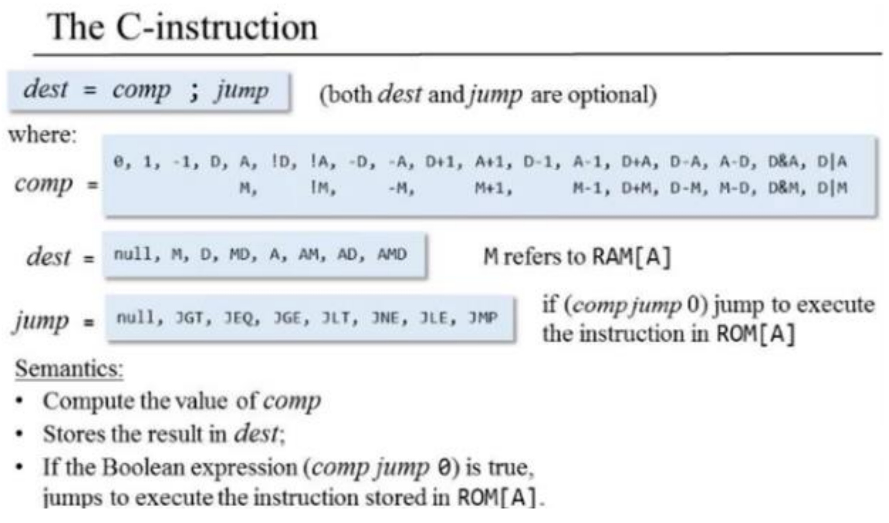
Syntax: `@value`

Where *value* is either:
- a non-negative decimal constant or
- a symbol referring to such a constant (later)

Semantics:
- Sets the A register to *value*
- Side effect: RAM[A] becomes the selected RAM register

Example: `@21`

Effect:
- Sets the A register to 21
- RAM[21] becomes the selected RAM register



The C-instruction

dest = *comp* ; *jump* (both *dest* and *jump* are optional)

where:

comp =	0, 1, -1, D, A, !D, !A, -D, -A, D+1, A+1, D-1, A-1, D+A, D-A, A-D, D&A, D|A M, !M, -M, M+1, M-1, D+M, D-M, M-D, D&M, D|M
dest =	null, M, D, MD, A, AM, AD, AMD — M refers to RAM[A]
jump =	null, JGT, JEQ, JGE, JLT, JNE, JLE, JMP — if (*comp jump* 0) jump to execute the instruction in ROM[A]

Semantics:
- Compute the value of *comp*
- Stores the result in *dest*;
- If the Boolean expression (*comp jump* 0) is true, jumps to execute the instruction stored in ROM[A].

Translate the following into Assembly Instructions:

1) Set RAM[0] to 3 Set RAM[1] to 5 Set RAM[2] to 1 Set RAM[3] to -1	MOV RAM[0], 3 MOV RAM[1], 5 MOV RAM[2], 1 MOV RAM[3], -1
2) Set RAM[0] to 2 Set RAM[1] to 3 Set RAM[2] = RAM[0] + RAM[1]	MOV RAM[0], 2 MOV RAM[1], 3 MOV RAM[2], RAM[1], RAM[2]
3) Set **D** to **A - 1**	SUB A, 1 MOV D, A
4) Set both **A** and **D** to **A + 1**	ADD A, 1 MOV D, A
5) Set **D** to **19**	MOV D, 19

6) Set both **A** and **D** to **A + D**	`ADD A, D`
	`MOV D, A`

7) Set **RAM[5034]** to **D - 1**	`SUB D, 1`
	`MOV RAM[5034], D`

8) Set **RAM[543]** to **171**	`MOV RAM[543], 171`

9) Increment **RAM[7]** by **1** and store result in **D**	`MOV D, RAM[7]`
	`ADD D, 1`
	`MOV RAM[7], D`

10) Increment **RAM[12]** by **3** and store result in **D**	`MOV D, RAM[12]`
	`ADD D, 3`
	`MOV RAM[12], D`

11) // Convert the following Java code to assembly
`int i = 5;`
`i++;`
`i+=2;`
`i-=3;`
	`i: .word 5`
	
	`// inc i`
	`LOAD i`
	`ADD 1`
	`STORE i`
	
	`LOAD i`
	`ADD 2`
	`STORE i`
	
	`LOAD i`
	`SUB 3`
	`STORE i`

12) // Convert the following Java code to assembly
`int i = 5;`
`int j = 10;`
`int k = i - j;`
	`i: .word 5`
	`j: .word 10`
	`k: .word 0`
	
	`LOAD i`
	`SUB j`
	`STORE k`

Translate the following tasks into Assembly Instructions

1) **`sum = 0`**	`sum: .word 0`
	
	`MOV sum, 0`

2) **`j = j + 1`**	`LOAD j`
	`ADD 1`
	`STORE j`

3) **`q = sum + 12 - j`**	`LOAD q`
	`ADD 12`
	`SUB j`
	`STORE q`

4) // Declare that arr=100 and n =10
`arr[3] = -1`
	`arr: .word 100`
	`n = .word 10`
	
	`// R1 = register 1`
	`MOV R1, 3`
	`ADD R1, R1, R1`
	`ADD R1, R1, arr`
	`MOV RAM[R1], -1`

5) **`arr[j] = 0`**	`// set array j to 0`
	
	`LOAD j`
	`ADD j, j, j`
	`ADD j, arr, j`
	`MOV RAM[j], 0`

6) **`arr[j] = 17`**	`LOAD j`
	`ADD j, j, j`
	`ADD j, arr, j`
	`MOV RAM[j], 17`

Lab #5 - Machine Language Jumps

Translate the following instructions into Assembly Instructions

1) **`goto 50`**	`JMP 50`

2) **`if D==0 goto 112`**	`CMP D, 0`
	`JE 112`

3) **`if D<9 goto 507`**	`CMP D, 9`
	`JL 507`

4) **`if RAM[12]>0 goto 50`**	`LOAD R1, RAM[12]`
	`CMP R1`
	`JG 50`

5) **`if sum>0 goto END`**	`LOAD R1, sum`
	`CMP R1, 0`
	`JG END`

6) **`if x[i]<=0 goto NEXT`**	`LOAD R1, x`
	`ADD R1, R1, i`
	`LOAD R2, RAM[R1]`
	`CMP R2, 0`
	`JLE NEXT`

Lab #5 - Machine Language Loops

Translate the following instructions into Assembly Instructions

1)
`int n = 5;`
`for (int i=1;i<=n;i++) {}`
	`n: .word 5`
	`i: .word 1`
	
	`MOV R1, n`
	`MOV R2, i`
	
	`START:`
	
	`ADD R2, R2, 1`
	
	`CMP R2, R1`
	
	`JLE START`
	
	`END:`

2)
`int sum = 0;`
`int n = 5;`
`for (int i=1;i<=n;i++) {`
`  sum += i;`
`}`
	`sum: .word 0`
	`n: .word 5`
	`i: .word 1`
	
	`MOV R1, sum`
	`MOV R2, n`
	`MOV R3, i`
	
	`START:`
	
	`ADD R1, R2, R3`
	
	`ADD R3, R3, 1`
	
	`CMP R3, R2`
	
	`JLE START`
	
	`END:`

3)

```
// Declare an arr at 100
// Size (n) of 10
for (int i=0; i<n; i++)
   arr[i] = -1;
```

```
arr: .space 40
n: .word 10
i: .word

MOV R1, arr
MOV R2, n
MOV R3, i

START:

MOV RAM[R1], -1
ADD R3, R3, 1
ADD R1, R1, R3, LSL #2

CMP R3, R2

JL START

END:
```

4)

```
// Declare an arr at 50
// Size (n) of 5
for (int i=0; i<n; i++)
   arr[i] = 100;
```

```
arr: .space 20
n: .word 5
i: .word 0

MOV R1, arr
MOV R2, n
MOV R3, i

START:
MOV RAM[R1], 100
ADD R3, R3, 1

ADD R1, R1, R3, LSL #2
CMP R3, R2
JL START

END:
```