Computer Organization

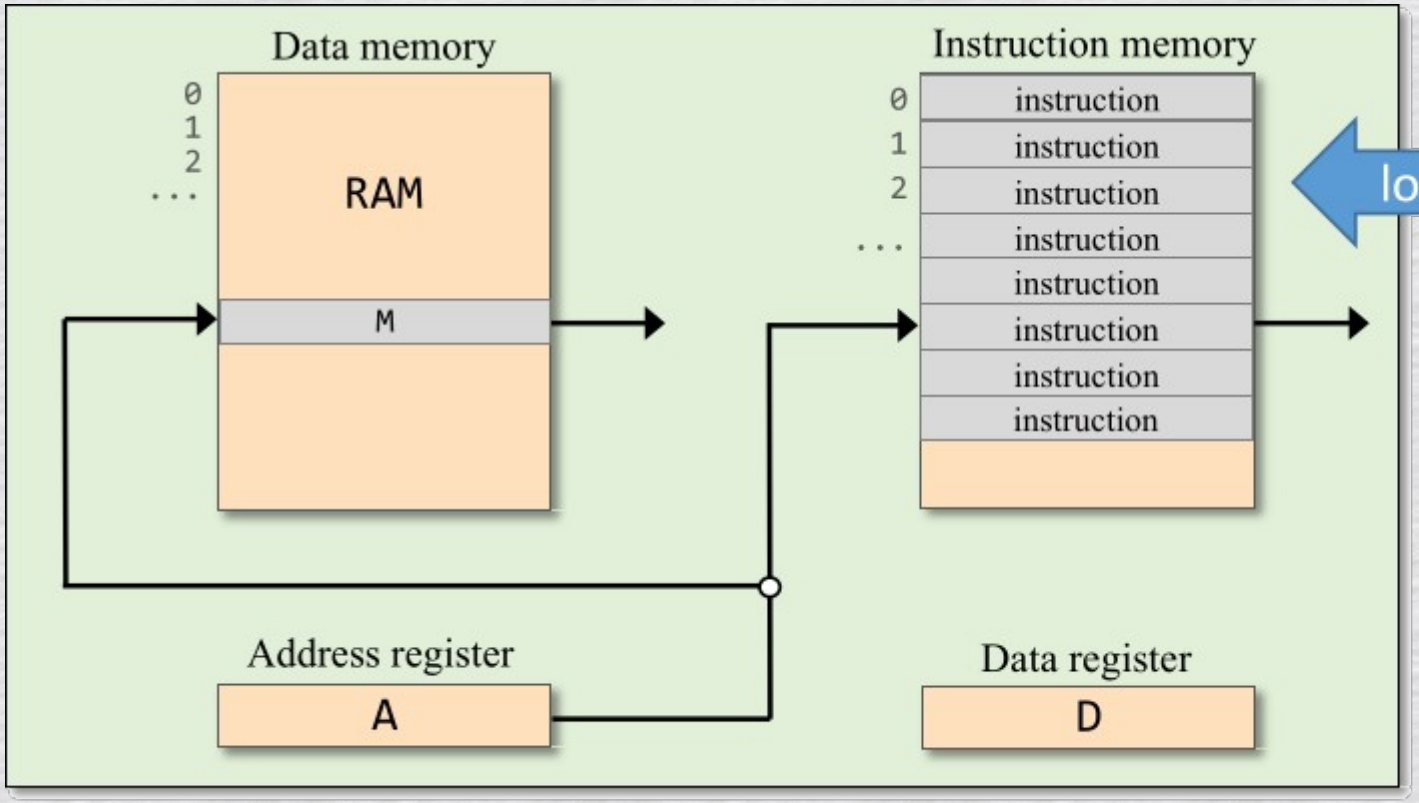
Branch HACK Instructions

Branching

- The HACK computer typically follows an iterative process regarding the progression of instructions read in via a HACK assembly file

- The only time this deviates is when branch instructions are involved
  - These are typically a combination of A and C instructions

Assembly Instructions



Convention:

The first instruction is loaded into address 0, the next instruction into address 1, and so on.

Unconditional & Conditional Branches

Unconditional branching example (pseudocode)

```
0   instruction
1   instruction
2   instruction
3   instruction
4   goto 7
5   instruction
6   instruction
7   instruction
8   instruction
9   goto 2
10  instruction
11  ...
```

Flow of control:

0,1,2,3,4,

7,8,9,

2,3,4,

7,8,9,

2,3,4,

...

Conditional branching example (pseudocode)

```
0   instruction
1   instruction
2   instruction
3   instruction
4   if (condition) goto 7
5   instruction
6   instruction
7   instruction
8   instruction
9   instruction
... ...
```

Flow of control:

0,1,2,3,4,

if *condition* is true

7,8,9,...

else

5,6,7,8,9,...

Branching – Instructions

Example (Pseudocode):

```
0    instruction
1    instruction
2    if (D>0) goto 6
3    instruction
4    instruction
5    instruction
6    instruction
7    instruction
...  ...
```

In Hack:

```
...
// if (D>0) goto 6
@6
D;JGT
...
```

Typical branching instructions:

```
D;JGT  // if D > 0 jump
D;JGE  // if D ≥ 0 jump
D;JLT  // if D < 0 jump
D;JLE  // if D ≤ 0 jump
D;JEQ  // if D = 0 jump
D;JNE  // if D ≠ 0 jump
0;JMP  // jump
```

} to the instruction stored in ROM[A]

Branching – cont.

```
D;JGT  // if D > 0 jump
D;JGE  // if D ≥ 0 jump
D;JLT  // if D < 0 jump
D;JLE  // if D ≤ 0 jump
D;JEQ  // if D = 0 jump
D;JNE  // if D ≠ 0 jump
0;JMP  // jump
```

} to the instruction stored in ROM[A]

```
// if (D = 0) goto 300
?
```

`@constant` (A ← constant)

```
A=1
D=-1
M=0
...
```

```
A=M
D=A
M=D
...
```

```
D=D-A
A=A-1
M=D+1
...
```

Branching – cont.

```
D;JGT  // if D > 0 jump
D;JGE  // if D ≥ 0 jump
D;JLT  // if D < 0 jump
D;JLE  // if D ≤ 0 jump
D;JEQ  // if D = 0 jump
D;JNE  // if D ≠ 0 jump
0;JMP  // jump
```

to the instruction stored in ROM[A]

```
// if (D = 0) goto 300
@300
D;JEQ
```

@*constant* (A←*constant*)

```
A=1
D=-1
M=0
...
```

```
A=M
D=A
M=D
...
```

```
D=D-A
A=A-1
M=D+1
...
```

Branching – cont.

```
D;JGT  // if D > 0 jump
D;JGE  // if D ≥ 0 jump
D;JLT  // if D < 0 jump
D;JLE  // if D ≤ 0 jump
D;JEQ  // if D = 0 jump
D;JNE  // if D ≠ 0 jump
0;JMP  // jump
```

to the instruction stored in ROM[A]

@*constant* (A←*constant*)

```
A=1
D=-1
M=0
...
```

```
A=M
D=A
M=D
...
```

```
D=D-A
A=A-1
M=D+1
...
```

```
// if (RAM[3] < 100) goto 12
?
```

Branching – cont.

```
D;JGT  // if D > 0 jump
D;JGE  // if D ≥ 0 jump
D;JLT  // if D < 0 jump
D;JLE  // if D ≤ 0 jump
D;JEQ  // if D = 0 jump
D;JNE  // if D ≠ 0 jump
0;JMP  // jump
```

} to the instruction stored in ROM[A]

@*constant* (A←*constant*)

```
A=1
D=-1
M=0
...
```

```
A=M
D=A
M=D
...
```

```
D=D-A
A=A-1
M=D+1
...
```

```
// if (RAM[3] < 100) goto 12
// D = RAM[3] - 100
@3
D=M
@100
D=D-A
// if (D < 0) goto 12
@12
D;JLT
```