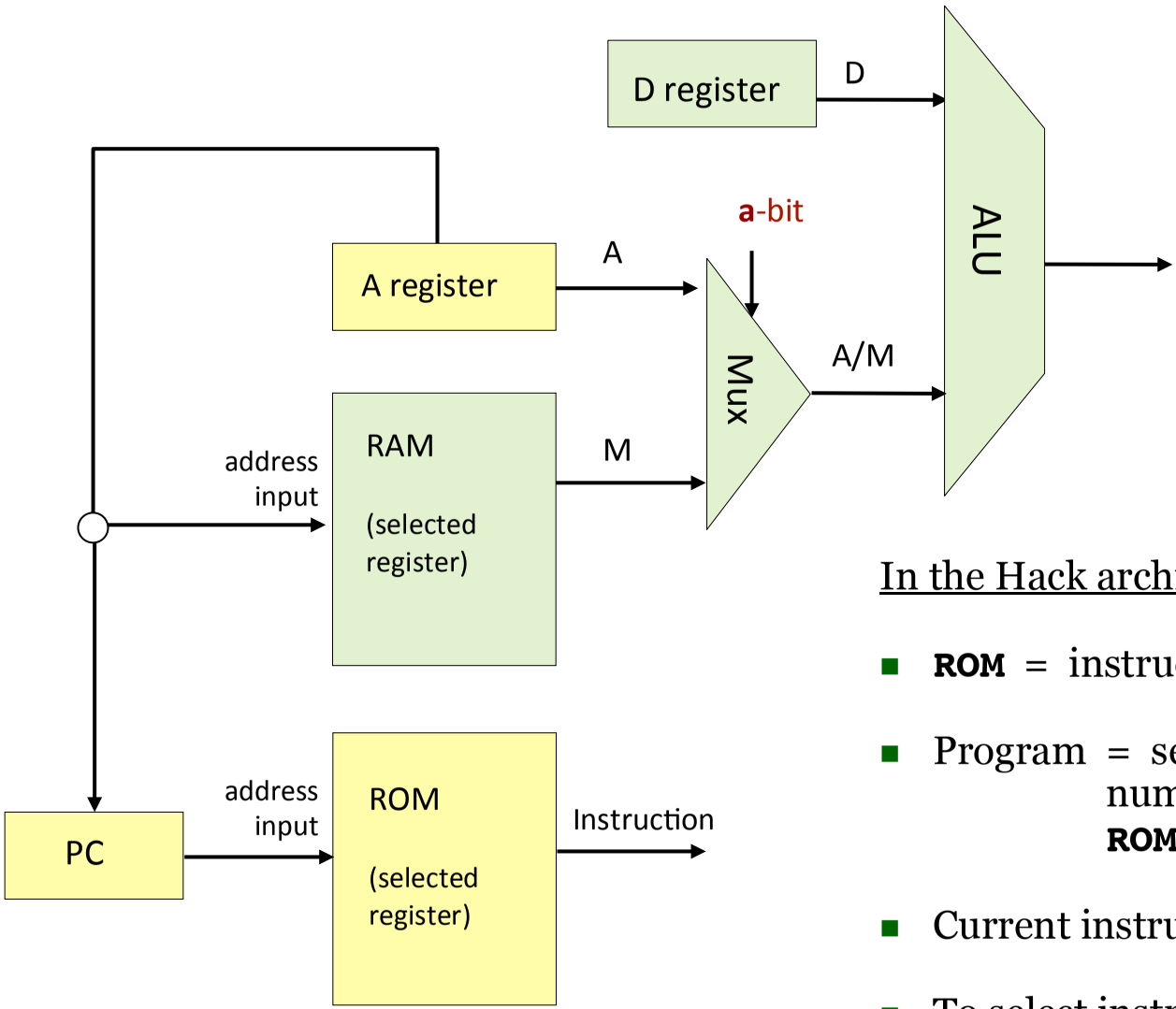
Control (focus on the yellow chips only)



In the Hack architecture:

- **ROM** = instruction memory
- Program = sequence of 16-bit numbers, starting at **ROM[0]**
- Current instruction = **ROM[PC]**
- To select instruction *n* from the **ROM**, we set **A** to *n*, using the instruction **@n**

C language

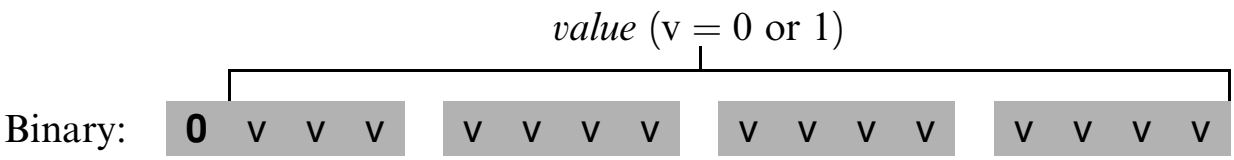
```
// Adds 1+...+100.
  int i = 1;
  int sum = 0;
  While (i <= 100){
     sum += i;
     i++;
  }
```

Hack machine language

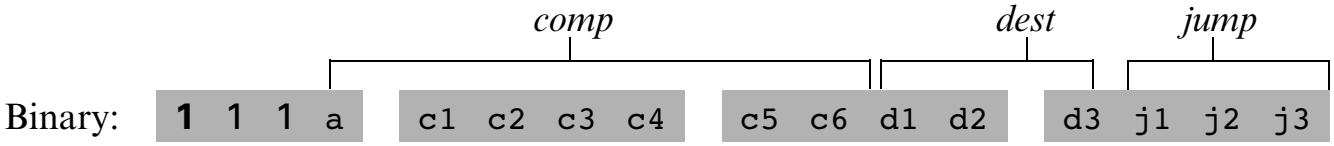
```
// Adds 1+...+100.
       @i     // i refers to some mem. location.
       M=1    // i=1
       @sum   // sum refers to some mem. location.
       M=0    // sum=0
 (LOOP)
       @i
       D=M    // D=i
       @100
       D=D-A  // D=i-100
       @END
       D;JGT  // If (i-100)>0 goto END
       @i
       D=M    // D=i
       @sum
       M=D+M  // sum=sum+i
       @i
       M=M+1  // i=i+1
       @LOOP
       0;JMP  // Goto LOOP
  (END)
       @END
       0;JMP  // Infinite loop
```

Figure 4.2 C and assembly versions of the same program. The infinite loop at the program's end is our standard way to "terminate" the execution of Hack programs.

A-instruction: *@value* // Where *value* is either a non-negative decimal number
// or a symbol referring to such number.



C-instruction: *dest=comp;jump* // Either the *dest* or *jump* fields may be empty.
// If *dest* is empty, the "=" is omitted;
// If *jump* is empty, the ";" is omitted.



(when a=0) ***comp*** *mnemonic*	c1	c2	c3	c4	c5	c6	(when a=1) ***comp*** *mnemonic*
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	1	1	0	0	0	0	M
!D	0	0	1	1	0	1	
!A	1	1	0	0	0	1	!M
-D	0	0	1	1	1	1	
-A	1	1	0	0	1	1	-M
D+1	0	1	1	1	1	1	
A+1	1	1	0	1	1	1	M+1
D-1	0	0	1	1	1	0	
A-1	1	1	0	0	1	0	M-1
D+A	0	0	0	0	1	0	D+M
D-A	0	1	0	0	1	1	D-M
A-D	0	0	0	1	1	1	M-D
D&A	0	0	0	0	0	0	D&M
D|A	0	1	0	1	0	1	D|M

Figure 4.3 The *compute* field of the *C*-instruction. D and A are names of registers. M refers to the memory location addressed by A, namely, to Memory[A]. The symbols + and − denote 16-bit 2's complement addition and subtraction, while !, |, and & denote the 16-bit bit-wise Boolean operators Not, Or, and And, respectively. Note the similarity between this instruction set and the ALU specification given in figure 2.6.

d1	d2	d3	*Mnemonic*	*Destination (where to store the computed value)*
0	0	0	null	The value is not stored anywhere
0	0	1	M	Memory[A] (memory register addressed by A)
0	1	0	D	D register
0	1	1	MD	Memory[A] and D register
1	0	0	A	A register
1	0	1	AM	A register and Memory[A]
1	1	0	AD	A register and D register
1	1	1	AMD	A register, Memory[A], and D register

Figure 4.4 The *dest* field of the *C*-instruction.

j1 ($out < 0$)	**j2** ($out = 0$)	**j3** ($out > 0$)	**Mnemonic**	**Effect**
0	0	0	null	No jump
0	0	1	JGT	If $out > 0$ jump
0	1	0	JEQ	If $out = 0$ jump
0	1	1	JGE	If $out \geq 0$ jump
1	0	0	JLT	If $out < 0$ jump
1	0	1	JNE	If $out \neq 0$ jump
1	1	0	JLE	If $out \leq 0$ jump
1	1	1	JMP	Jump

Figure 4.5 The *jump* field of the *C*-instruction. *Out* refers to the ALU output (resulting from the instruction's *comp* part), and *jump* implies "continue execution with the instruction addressed by the A register."