Instruction Set Architecture -del processore Hack-

```
// Adds 1+...+100.
      @i     // i refers to some RAM location
      M=1    // i=1
      @sum   // sum refers to some RAM 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 += i
      @i
      M=M+1   // i++
      @LOOP
      0;JMP   // goto LOOP
 (END)
      @END
      0;JMP   // infinite loop
```

Prof. Ivan Lanese

Instruction Set Architecture

- Rappresenta l'interfaccia fra hardware e software
  - Abbiamo studiato i principali componenti di un processore (circuiti logici combinatori e sequenziali, memorie, microarchitettura,..)
  - Ora dobbiamo studiare come tali componenti vengono effettivamente usati
- Per usare un processore, lo si deve programmare tramite un suo specifico linguaggio, costituito da un insieme di istruzioni che solitamente viene detto "instruction set" che dipende dalla cosiddetta "instruction set architecture" (ISA)
- Studieremo nel dettaglio l'ISA del processore Hack
  - Una volta studiato, termineremo l'implementazione del processore Hack usando l'Hardware Description Language (HDL)
  - Il processore dovrà essere in grado di eseguire le istruzioni dell'ISA che studieremo

Versione binaria e simbolica di un ISA

- Le istruzioni eseguibili da un processore sono solitamente codificate da sequenze binarie
- Questa versione delle istruzioni viene detta versione "binaria"
- Per rendere più leggibili le istruzioni, esiste un modo simbolico di scrivere le medesime istruzioni. Ecco un esempio legato a Hack:

```
// Adds 1+...+100.
      @i     // i refers to some RAM location
      M=1    // i=1
      @sum   // sum refers to some RAM 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 += i
      @i
      M=M+1   // i++
      @LOOP
      0;JMP   // Goto LOOP

 (END)
      @END
      0;JMP   // Infinite loop
```

```
0000000000010000
1110111111001000
0000000000010001
1110101010001000
0000000000010000
1111110000010000
0000000001100100
1110010011010000
0000000000010010
1110001100000001
0000000000010000
1111110000010000
0000000000010001
1111000010001000
0000000000010000
1111110111001000
0000000000000100
1110101010000111
0000000000010010
1110101010000111
```

Il computer Hack

Hack è una macchina a 16-bit costituita da:

Memoria dati: **RAM** – una sequenza di registri a 16 bit che contiene dati

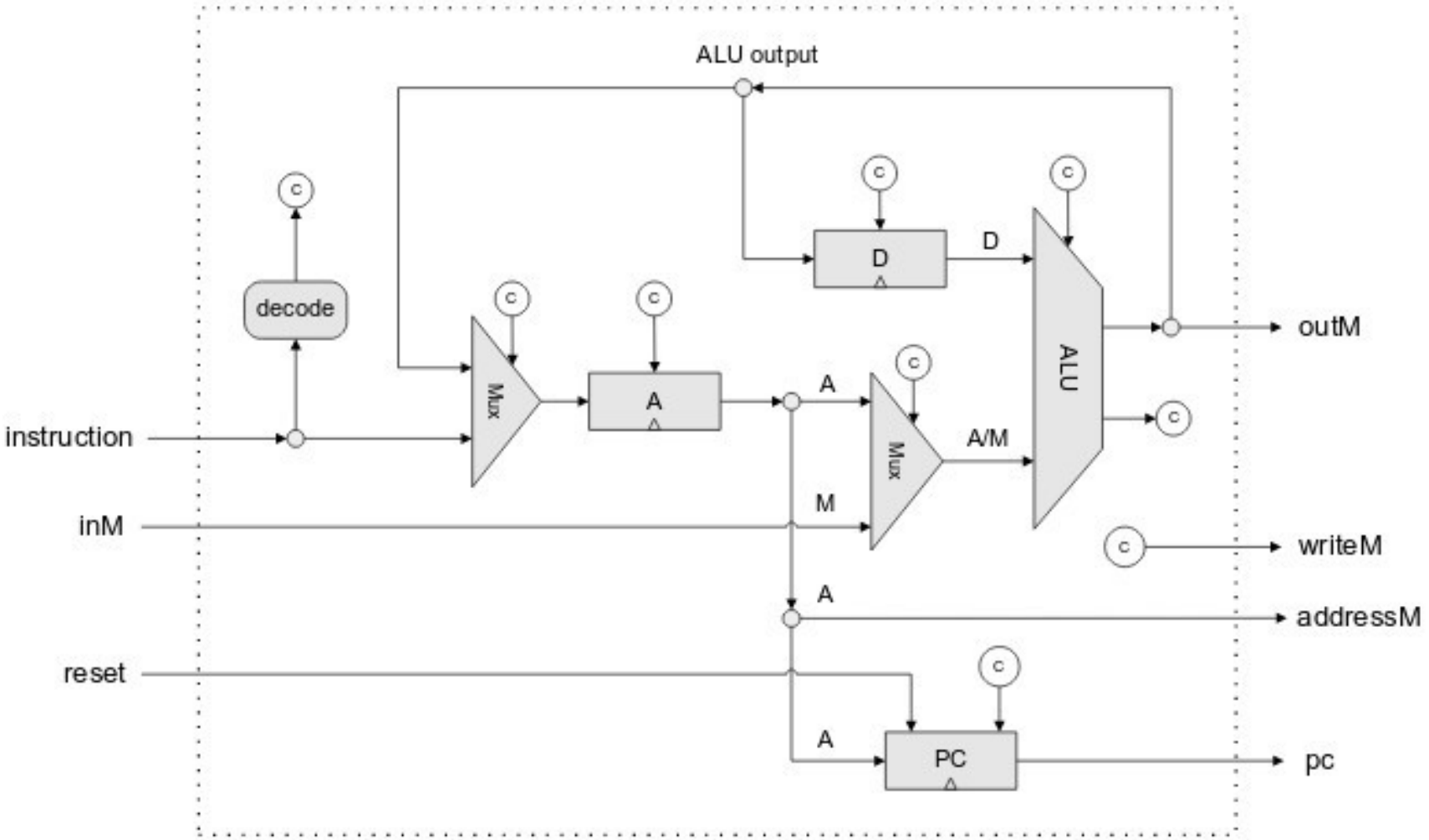
Memoria istruzioni: **ROM** – una sequenza di registri a 16 bit che contiene istruzioni da eseguire

Registri: **D**, **A** (registri interni del processore),
M (registro attualmente puntato da A in memoria RAM, **RAM[A]**)

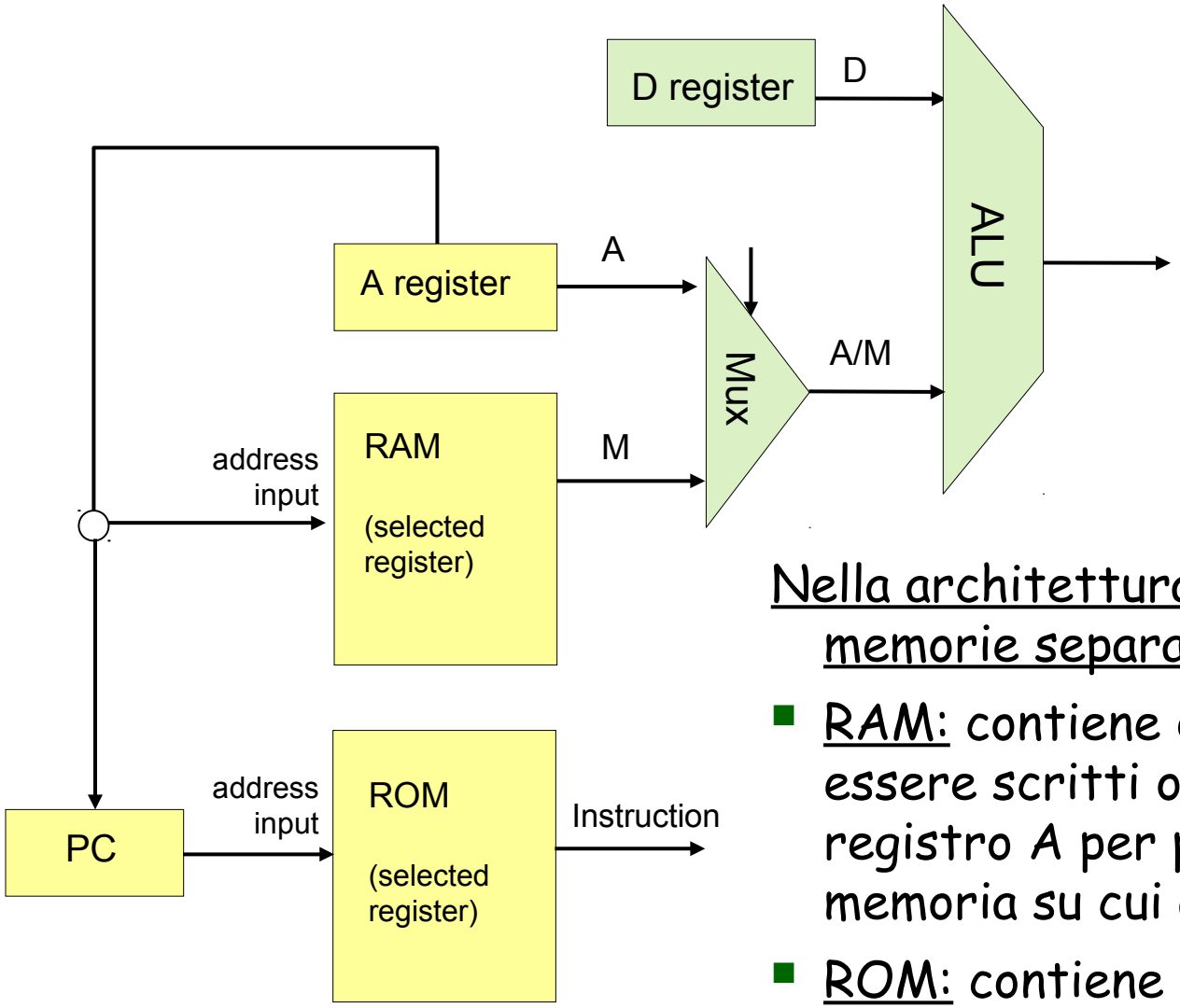
Elaborazione: **ALU**, la conosciamo già

Program counter: **PC**, contiene l'indirizzo della prossima istruzione da eseguire (**ROM[PC]**). Inizialmente **PC** vale 0
(i programmi da eseguire inizieranno quindi da ROM[0])

Architettura del processore Hack



RAM per dati e ROM per istruzioni



Nella architettura Hack, esistono due memorie separate:

- RAM: contiene dati che possono essere scritti o letti (si usa il registro A per puntare alla cella di memoria su cui agire)
- ROM: contiene le istruzioni da eseguire, il registro PC indica quale istruzione deve essere letta

Tipi di istruzioni

- Esistono due tipi di istruzioni:
  - A-instruction: istruzioni che si limitano a caricare un valore all'interno del registro A
    - È possibile caricare una costante numerica non negativa (ad esempio, 5, 7, 12453,…), eventualmente rappresentata da un nome (vedremo più avanti come si usano nomi per rappresentare numeri)
  - C-instruction: istruzioni che eseguono una computazione prelevando i due operandi dai registri A, D, M, oppure le costanti 0, 1 o -1, e memorizzano il risultato in A, D, M o combinazioni di questi
    - Le computazioni che si possono eseguire sono:
      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
    - D e A sono i registri interni, M coincide con RAM[A]

Le A-instruction

```
@value          // A ← value
```

Dove *value* è o un numero o un nome che rappresenta un numero

Esempi:

- Caricare una costante
  (D = **value**)

```
@17     // A = 17
D = A   // D = 17
```

- Caricare un valore da **RAM**
  (D = **RAM[A]**)

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

- Selezionare una locazione **ROM**
  (**PC = A**)

```
@17     // A = 17
0;JMP   // mette 17 in PC, la prossima
        // istruzione sara' ROM[17]
```

Le C-instruction

C-command: dest = comp ; jump ("dest =" e "; jump" sono opzionali)

dove:

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 = M , D , MD , A , AM , AD , AMD, o nullo (in questo caso viene omesso)

jump = JGT , JEQ , JGE , JLT , JNE , JLE , JMP, o nullo (omesso)

Nel comando "dest = comp; jump", il salto si esegue controllando ciò che esce dalla ALU confrontandolo con 0 (tramite i suoi bit di stato). Per esempio, in "D=D+1;JLT", si salta se D+1 < 0.
Si salta all'indirizzo contenuto nel registra A.

Convenzione per costanti Booleane: "true" si rappresenta con -1 (tutti i bit a 1), "false" con 0 (tutti i bit a 0)

Iniziamo a programmare Hack

- Due modi per mettere in RAM[2] la somma RAM[0]+RAM[1]

```
// RAM[2]=RAM[0]+RAM[1]

D=0    // D=0
@0
D=D+M // D += RAM[0]
@1
D=D+M // D += RAM[1]
@2
M=D    // RAM[2]=D
```

```
// RAM[2]=RAM[0]+RAM[1]

@2
M=0    // RAM[2]=0
@0
D=M    // D=RAM[0]
@2
M=D+M // RAM[2]+= D
@1
D=M    // D=RAM[1]
@2
M=D+M // RAM[2]+= D
```

Iniziamo a programmare Hack

- RAM[2] = RAM[0] × RAM[1] (assumendo RAM[1] diverso da 0)

```
// RAM[2] = RAM[0] × RAM[1]

   @2
   M=0
(LOOP)
   @0
   D=M
   @2
   M=D+M
   @1
   MD=M-1
   @LOOP
   D;JGT
```

Dichiarazione di etichetta:
si usa il simbolo "LOOP" per far riferimento all'indirizzo dell'istruzione successiva al punto di dichiarazione

Uso di etichetta:
il simbolo "LOOP" sarà sostituito con il valore dell'etichetta, saltando così al punto di sua dichiarazione

Iniziamo a programmare Hack

- RAM[2] = RAM[0] × RAM[1] (assumendo RAM[1] diverso da 0)

```
// RAM[2] = RAM[0] × RAM[1]

   @2
   M=0          // RAM[2]=0
(LOOP)          // do
   @0
   D=M          //   D=RAM[0]
   @2
   M=D+M        //   RAM[2] += D
   @1
   MD=M-1       //   D = --RAM[1]
   @LOOP
   D;JGT        // while D>0
```

Iniziamo a programmare Hack

- RAM[0..9] = [10, 9, 8,…, 1]

```
// RAM[0..9]=[10, 9, 8,.., 1]
   @10
   M=-1          // RAM[10]=-1   variable for address
   @10
   D=A
   @11
   M=D           // RAM[11]=10   variable for value
(LOOP)           // do
   @11
   D=M
   M=M-1         //  D=RAM[11]--
   @10
   AM=M+1        //  A = ++RAM[10]
   M=D           //  RAM[A] = D
   @11
   D=M
   @LOOP
   D;JGT         // while RAM[11]>0
```

Iniziamo a programmare Hack

- RAM[10] = RAM[9]+RAM[8]+…+RAM[1]+RAM[0]

```
// RAM[10]=RAM[9]+RAM[8]+…+RAM[1]+RAM[0]
   @10
   M=0          // RAM[10]=0
   @9
   D=A
   @11
   M=D          // RAM[11]=9
(LOOP)          // do
   @11
   A=M
   D=M          //   D=RAM[RAM[11]]
   @10
   M=D+M        //   RAM[10] += D
   @11
   MD=M-1       //   D = --RAM[11]
   @LOOP        // while D>=0
   D;JGE
(END)           // final loop
   @END
   0;JMP
```

Indica che il programma è terminato

Iniziamo a programmare Hack

- RAM[2] = max(RAM[0],RAM[1])

```
// RAM[2]=max(RAM[0],RAM[1])

   @0
   D=M          // D=RAM[0]
   @1
   D=D-M        // D -= RAM[1]
   @ELSE
   D;JLT        // jump if(D<0)
   @0
   D=M
   @2
   M=D          //   RAM[2]=RAM[0]
   @END
   0;JMP        // jump to END
```

```
(ELSE) //else
   @1
   D=M
   @2
   M=D //   RAM[2]=RAM[1]
(END)  //final loop
   @END
   0;JMP
```

Iniziamo a programmare Hack

- RAM[10] = max(RAM[9],RAM[8],…,RAM[1],RAM[0])

```
// RAM[10] = max(RAM[9..0])

   @9
   D=M  // D=RAM[9]
   @10
   M=D  // RAM[10]=D
   @8
   D=A
   @11
   M=D  // RAM[11]=8
(LOOP)  // do
   @11
   A=M
   D=M  //  D=RAM[RAM[11]]
   @10
   D=D-M // D -= RAM[10]
   @NEXT
   D;JLE  //  jump if (D <= 0)
   @11
   A=M
   D=M
   @10
   M=D    //    RAM[10]=RAM[RAM[11]]
(NEXT)    //  endif
   @11
   MD=M-1 //  RAM[11]--
   @LOOP
   D;JGE   // while RAM[11]>=0
(END)     // final loop
   @END
   0;JMP
```

Esercizi

- Scrivere dei programmi HACK che rispondano alle specifiche seguenti
- RAM[2] = RAM[1] –RAM[0] - 2
- RAM[2] = RAM[1] nand RAM[0] //nand bit a bit
- Mettere il valore 1 in tutte le celle di memoria con indirizzo compreso tra RAM[0] e RAM[1] assumendo RAM[1] > RAM[0] > 1
- RAM[2] = RAM[1] * (2^RAM[0])
- RAM[2] = RAM[1] / RAM[0], RAM[3] = RAM[1] mod RAM[0] //divisione intera

Regole generali per controllo del flusso: costrutto “if”

Alto livello:

```
if condition {
   codice block 1}
else {
   codice block 2}
code block 3
```

Livello Hack:

```
   D ← condition
   @IF_FALSE
   D;JEQ
   codice block 1
   @END
   0;JMP
(IF_FALSE)
   codice block 2
(END)
   codice block 3
```

Regole generali per controllo del flusso: costrutto “while”

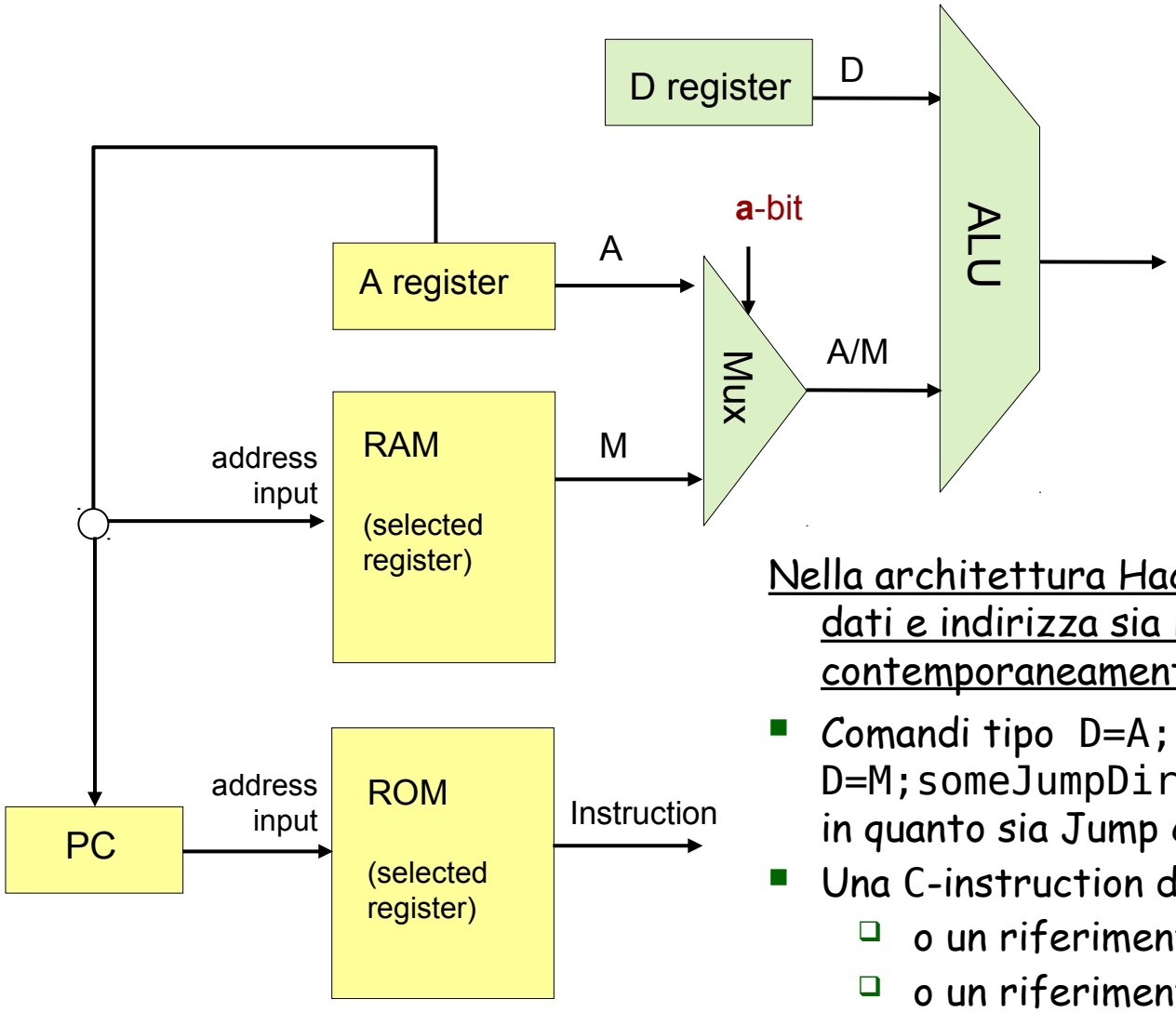
Alto livello:

```
while condition {
    codice block 1
}
codice block 2
```

Livello Hack:

```
  (LOOP)
       D ← condition
       @END
       D;JEQ
       codice block 1
       @LOOP
       0;JMP
(END)
       codice block 2
```

Annotazione sull'uso del registro A per puntare alle memorie



Nella architettura Hack, il registro A contiene dati e indirizza sia la RAM che la ROM, contemporaneamente. Quindi:

- Comandi tipo `D=A;someJumpDirective` o `D=M;someJumpDirective` non hanno senso in quanto sia Jump che M usano A
- Una C-instruction dovrebbe contenere
  - o un riferimento a M,
  - o un riferimento a A,
  - o una direttiva jump,

  ma non 2 di questi

Uso dei simboli nel linguaggio Hack

In Hack si usano due tipi di simboli:

- Etichette: Usate per fare riferimento ad indirizzi della ROM. Dichiarate tramite direttiva **(XXX)** che definisce il simbolo **XXX** che farà riferimento all'indirizzo di ROM dell'istruzione successiva alla dichiarazione
- Variabili: Usate per far riferimento ad indirizzi in memoria RAM. Ci sono due tipi di variabili:
  - **pre-definite**: ad esempio **SCREEN** e **KBD** che fanno riferimento agli indirizzi RAM 16384 e 24576 (indicano rispettivamente l'inizio della memoria per gestire lo schermo e la locazione dove viene inserito il codice di un eventuale tasto premuto su tastiera)
  - **definite dall'utente**: ogni simbolo non predefinito **xxx** che appare in un programma Hack senza essere dichiarato usando **(xxx)** viene trattato come una variabile, e gli viene assegnato in automatico un valore (a partire dal valore 16, da 0 a 15 sono usati dalle variabili predefinite **R0**..**R15**)

Domanda: Chi assegna i valori ai simboli?

Risposta: L' *assemblatore*, che è il programma che traduce un programma Hack nel codice binario (studieremo più avanti)

```
// Typical symbolic
// Hack code, meaning
// not important
   @R0
   D=M
   @INFINITE_LOOP
   D;JLE
   @counter
   M=D
   @SCREEN
   D=A
   @addr
   M=D
(LOOP)
   @addr
   A=M
   M=-1
   @addr
   D=M
   @32
   D=D+A
   @addr
   M=D
   @counter
   MD=M-1
   @LOOP
   D;JGT
(INFINITE_LOOP)
   @INFINITE_LOOP
   0;JMP
```

Chiudiamo con un esempio di un ipotetico programma C

Linguaggio C:

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

Convenzioni:

- Variabili: minuscolo
- Etichette: maiuscolo
- Comandi: maiuscolo

Linguaggio Hack:

```
// Adds 1+...+100.
       @i      // i refers to some RAM location
       M=1     // i=1
       @sum    // sum refers to some RAM 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 += i
       @i
       M=M+1   // i++
       @LOOP
       0;JMP   // goto LOOP
 (END)
       @END
       0;JMP   // Infinite loop
```

Sommario

- Hack è un semplice linguaggio machina
- Ha una sintassi "user-friendly": `D=D+A` invece di `ADD D,D,A`
- I comandi Hack possono prelevare dalla memoria al più un operando
  - ogni operazione che richiede di operare sulla RAM deve prima caricare l'indirizzo di memoria del dato da usare nel registro A tramite una **A**-instruction e successivamente una C-instruction può accedere a tale operando usando "M"
- L' assemblatore per il linguaggio Hack sarà discusso più avanti; prima procederemo alla realizzazione del processore e dell'intera architettura del nostro computer