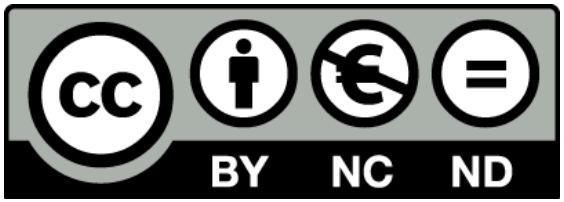
Laboratorio di Programmazione di Sistema

Controllo dell'Esecuzione nei Linguaggi Assembly

Luca Forlizzi, Ph.D.

Versione 20.2



Concetti di Base

- Nei linguaggi *ASM* non esistono istruzioni che contengono al loro interno altre istruzioni
- Quindi gli *ASM* non hanno istruzioni di selezione o di ciclo che contengono altre istruzioni
- Nei linguaggi *ASM*, il controllo del flusso viene realizzato in maniera simile a quanto accade in Unstructured-C

Concetti di Base

- Una abstract machine M che si trova in un modo di funzionamento di tipo RUN, dopo aver eseguito un'istruzione, ne esegue immediatamente un'altra
- Chiamiamo *ordine di esecuzione*, l'ordine in cui le istruzioni di un programma vengono eseguite
- Per la maggior parte delle istruzioni, l'istruzione successiva in ordine di esecuzione coincide con l'istruzione successiva in ordine testuale
- Le *istruzioni di salto* sono quelle per cui, invece, l'istruzione successiva in ordine di esecuzione può non essere, e in certi casi non è mai, l'istruzione successiva in ordine testuale

Istruzioni di Salto

- Tipici nomi simbolici per le istruzioni di salto sono abbreviazioni delle parole inglesi *jump* o *branch*
- La semantica di un'istruzione di salto I individua un'istruzione K del programma chiamata *istruzione di destinazione del salto*
- Un'istruzione di salto I ha *destinazione statica* se la propria istruzione di destinazione del salto è la stessa, ogni volta che viene eseguita I
- Un'istruzione di salto I ha *destinazione dinamica* se, ogni volta che viene eseguita, individua la propria istruzione di destinazione del salto (che può quindi variare ad ogni esecuzione)

Istruzioni di Salto

- In quasi tutti gli *ASM*, nel caso di istruzioni di salto con destinazione statica, si può indicare l'istruzione di destinazione del salto per mezzo di una label
- Come sappiamo, una definizione di label è un costrutto che può essere posto all'inizio di una riga del codice sorgente
- Una label è un identificatore scelto dal programmatore che permette di "dare un nome" al costrutto *ASM* che, nel codice sorgente, segue immediatamente la definizione della label
- In particolare, se il costrutto che segue la definizione di una label L è un'istruzione K, un'istruzione di salto che voglia indicare K come istruzione di destinazione del salto, lo può fare usando L

Istruzioni di Salto

- In quasi tutti gli *ASM*, le label si definiscono esattamente come in linguaggio C
- All'inizio di una riga si scrive il nome della label seguito (senza spazi in mezzo) da `:`
- La label identifica il costrutto che, in ordine testuale, segue immediatamente (dopo un qualunque numero di spazi o righe vuote) la definizione
- Ad esempio, in un *ASM* per la *ISA* Intel 8086, se si vuole definire la label `st1` per identificare l'istruzione `mov bx,cx`, si può scrivere

```
st1:  mov bx,cx
```

- In modo equivalente si può scrivere

```
st1:
mov bx,cx
```

Istruzioni di Salto

- Un'istruzione di *salto incondizionato I*, ogni volta che viene eseguita, fa sì che la abstract machine effettui "un salto" nell'ordine di esecuzione
- L'istruzione successiva di *I* nell'ordine di esecuzione sarà, in ciascun caso, l'istruzione di destinazione del salto
- Un'istruzione di salto incondizionato che determina staticamente l'istruzione di destinazione del salto, è l'analogo dell'istruzione `goto` in *UC*

Istruzioni di Salto

- Un'istruzione di *salto condizionato*, invece, non sempre fa sì che la abstract machine effettui "un salto" nell'ordine di esecuzione
- Ogni volta che viene eseguita, un'istruzione di salto condizionato *I* decide, in base ad una condizione, quale debba essere l'istruzione successiva nell'ordine di esecuzione, tra le due seguenti possibilità
  - l'istruzione di destinazione del salto
  - l'istruzione successiva di *I* nell'ordine testuale
- Se una determinata condizione, indicata da *I*, risulta vera, allora l'istruzione successiva di *I* nell'ordine di esecuzione sarà l'istruzione di destinazione del salto
- Altrimenti l'istruzione successiva di *I* nell'ordine di esecuzione sarà l'istruzione successiva di *I* nell'ordine testuale

Istruzioni di Salto

- La seguente tabella riassume le tipologie di istruzioni di salto

Istruzioni di salto	**destinazione statica**	**destinazione dinamica**
incondizionato	ad ogni esecuzione salta, la destinazione del salto è sempre la stessa	ad ogni esecuzione salta e determina la destinazione del salto
condizionato	ad ogni esecuzione decide se saltare, la destinazione del salto è sempre la stessa	ad ogni esecuzione decide se saltare, e inoltre determina la destinazione del salto

- Questa presentazione tratta solo istruzioni di salto con destinazione statica

Condizioni

- Consideriamo un programma in esecuzione; indichiamo con t_0 l'istante di tempo in cui il programma è stato avviato e con $t > t_0$ un generico istante di tempo durante il quale il programma è in esecuzione
- Una *condizione* è un valore booleano che esprime, ad ogni istante t, la verità o la falsità di un determinato fatto relativo allo *stato della computazione*, ovvero
  - un fatto relativo ai valori che le variabili del programma hanno al tempo t
  - oppure un fatto, determinato dalle istruzioni eseguite dall'istante t_0 fino all'istante t, che ha un effetto sui risultati dei calcoli eseguiti fino all'istante t

Condizioni

- Nei linguaggi *ASM*, vi sono istruzioni il cui effetto dipende da una determinata condizione
- In altre parole, tali istruzioni decidono quale effetto produrre anche in base al fatto che una determinata condizione sia vera oppure falsa
- L'esempio più significativo è quello delle istruzioni di salto condizionato, che decidono se saltare o meno in base a una condizione

Condizioni

- Anche negli *HLL* vi sono istruzioni che prendono decisioni sulla base di condizioni
- Ad esempio, un'istruzione iterativa come
  `while ( EC ) S;`
  dove `EC` è un'espressione di controllo e `S` è un'altra istruzione, decide se iterare `S` o meno sulla base del valore di `EC`
- In questo caso, quindi, la condizione è il valore di verità o falsità di `EC`

Condizioni

- Negli *HLL*, tutte le istruzioni di selezione e iterazione effettuano due operazioni
  1. Determinano la condizione, ovvero il valore booleano del fatto associato alla condizione
  2. Decidono, sulla base del valore determinato, come prosegue la computazione
- Tali due operazioni vengono sempre effettuate come un'unica, indivisibile, istruzione
- Ciò accade anche in *UC*: ad esempio l'istruzione
  `if ( x > y ) goto L;`
  effettua il confronto tra `x` e `y` e prende la decisione se saltare o meno

Condizioni

- L'approccio tipico degli *ASM*, è invece quello di separare la determinazione della condizione dalla decisione su come proseguire la computazione
  1. Alcune istruzioni determinano condizioni e le memorizzano in alcuni bit contenuti in registri di stato, detti *Condition Code Bits (CCB)*, in modo che possono essere utilizzate da istruzioni successive
  2. Altre istruzioni, dette *istruzioni condizionate*, leggono il contenuto dei CCB e, in base ad esso, effettuano determinate operazioni
- Negli *ASM* che adottano tale approccio, quindi, le istruzioni di salto condizionato sono esempi di istruzioni condizionate, in quanto decidono se effettuare o meno il salto in base al contenuto dei CCB

Condizioni

- Le istruzioni più comuni che determinano condizioni, sono le istruzioni aritmetiche e logiche
- In modo particolare, tra le istruzioni aritmetiche vi sono le istruzioni di confronto tra interi e le istruzioni di confronto tra valori floating point, che calcolano e memorizzano condizioni in base al confronto tra due numeri
- In alcuni *ASM-PM*, anche altre istruzioni determinano condizioni, ad esempio le istruzioni di trasferimento dati

Condizioni

- Il programmatore deve tener conto del fatto che se un'istruzione memorizza una condizione nei CCB, sovrascrive il precedente contenuto dei CCB, ovvero la condizione precedentemente determinata
- Supponiamo che si voglia fare in modo che:
  1. un'istruzione *I* determini una condizione **C**
  2. l'esito di un'istruzione condizionata *J* dipenda dal valore di **C** determinato da *I*
- È necessario che quando viene eseguita *J*, i CCB contengano ancora i valori determinati da *I*
- La soluzione più immediata è organizzare il codice in modo che dopo *I* e prima di *J* non vengano eseguite altre istruzioni che modifichino i CCB; molto spesso, ciò si ottiene scrivendo *J* come istruzione successiva ad *I* in ordine testuale

Condizioni

- Un'alternativa, possibile in alcuni *ASM-PM*, è
  - dopo aver eseguito *I*, copiare il contenuto dei CCB in un'altra parola *X*
  - prima di eseguire *J*, ripristinare i CCB al valore copiato in *X*
- Alcuni *ASM*, offrono anche istruzioni per calcolare e memorizzare il risultato di un'espressione relazionale, che sono utili per evitare di dover determinare più volte la stessa condizione; tali istruzioni sono analoghe alle istruzioni *UC*
  `V = A op B`
  dove `V` è una variabile, `A` e `B` sono una variabile o una costante e `op` è uno degli operatori relazionali (== , != , < , <= , > , >=)

Condizioni in MC68000

- MC68000 adotta l'approccio tipico della maggior parte degli *ASM*: il compito di determinare condizioni e quello di assumere decisioni vengono svolti da istruzioni differenti
- In MC68000, i 5 bit del registro di stato `sr` che hanno posizione compresa tra 0 e 4, sono usati per memorizzare condizioni determinate da operazioni su numeri interi
- Poiché i bit di posizione 5, 6 e 7 di `sr` sono inutilizzati, gli unici bit "utili" del byte formato dai bit di posizione tra 0 e 7 di `sr`, servono a memorizzare condizioni
- Pertanto il byte formato dai bit di posizione tra 0 e 7 di `sr` viene chiamato `ccr`, abbreviazione di *Condition Code Register* (impropriamente, in quanto non è un registro ma solo una parte di un registro)

Condizioni in MC68000

- È possibile trasferire il contenuto di `ccr` in un'altra parola, e viceversa
  - `move sr,D`, dove `D` è una parola memoria-ordinaria o un registro dati; copia il contenuto di `sr` in `D`; il formato del valore copiato è `word`, i bit di `sr` non utilizzati hanno valore 0
  - `move S,ccr`, dove `S` è un registro dati o una parola di memoria; copia il contenuto di `S` in `ccr`; il formato del valore copiato è (stranamente) `word`, ma solo i bit di posizione tra 0 e 4 vengono copiati nei corrispondenti bit di `ccr`
- Esiste anche un'istruzione `move S,sr` per modificare l'intero contenuto di `sr`, ma essa è disponibile solo nel modo di funzionamento *supervisor*, pertanto non può essere usata nei normali programmi che vengono eseguiti in modo *user*
- Ulteriori dettagli su `ccr` in una prossima lezione

Istruzioni di Confronto tra Interi

- Sono molte le istruzioni che effettuano un confronto tra valori interi e ne memorizzano l'esito in `ccr`
- Elenchiamo solo le più utilizzate
  - L'istruzione `cmp` ha due operandi, e ne confronta tra loro i rispettivi contenuti
  - L'istruzione `tst` confronta il contenuto del suo operando esplicito (un registro o una parola di memoria) con il valore 0
  - Molte istruzioni di trasferimento di dati confrontano il valore copiato con 0
  - Quasi tutte le istruzioni aritmetico-logiche confrontano il risultato prodotto con 0

Istruzioni di Confronto tra Interi

- L'istruzione `cmp` ha 2 operandi
  - Il primo operando
    - può essere un registro, una parola memoria-ordinaria o una parola memoria-immediata
    - il suo formato è quello indicato dall'estensione, se presente, oppure `word` in caso contrario
  - Il secondo operando
    - può essere un registro o una parola memoria-ordinaria
    - se è un registro indirizzi allora ha formato `long`, altrimenti ha lo stesso formato del primo operando
  - Se il secondo operando è una parola memoria-ordinaria, il primo operando deve essere una parola memoria-immediata
- `cmp` effettua il confronto tra il contenuto del secondo operando e quello del primo
- Ulteriori dettagli in una prossima lezione e in **[M68000]**

Istruzioni di Confronto tra Interi

- L'istruzione `tst` ha un operando
  - L'operando
    - può essere un registro dati o una parola memoria-ordinaria
    - il suo formato è quello indicato dall'estensione, se presente, oppure `word` in caso contrario
- `tst` effettua il confronto tra il contenuto dell'operando e il valore 0
- Ulteriori dettagli in una prossima lezione e in **[M68000]**

Istruzioni di Salto

- In MC68000-ASM1, le definizioni di label adottano la sintassi e la semantica usate da quasi tutti gli *ASM*, descritte in precedenza
- Le istruzioni di salto incondizionato sono `bra` (o `br`) e `jmp`
  - in `bra` ha destinazione statica
  - in `jmp` ha destinazione dinamica, la approfondiremo in future lezioni
- L'istruzione `bra` ha un solo operando, che può essere una label oppure un valore numerico
  - se l'operando è una label, l'istruzione di destinazione del salto è l'istruzione che segue la definizione della label
  - si tratterà in una futura lezione il caso in cui l'operando è un numero

Istruzioni di Salto

- Le istruzioni di salto condizionato, in MC68000-ASM1, hanno destinazione statica
- Esse hanno un solo operando che indica l'istruzione di destinazione del salto, per il quale valgono le stesse regole dell'operando di `bra`
- Il nome delle istruzioni di salto condizionato ha la forma `b`**cc**, dove **cc** è un codice di 2 caratteri che indica la specifica condizione controllata dall'istruzione

Istruzioni di Salto

- Nella tabella seguente sono indicati alcuni possibili valori del codice **cc**

Codice	Condizione	Operatore C corrispondente
`eq`	*Equal*	==
`ne`	*Not Equal*	!=
`lt`	*Less Than*	<
`le`	*Less Than or Equal*	<=
`gt`	*Greater Than*	>
`ge`	*Greater Than or Equal*	>=

- Ulteriori informazioni in una prossima lezione e in **[M68000]**

Istruzioni di Memorizzazione di Condizione

- In MC68000 sono disponibili istruzioni che copiano una specifica condizione memorizzata in `ccr` sotto forma di *flag* (*indicatore*)
- Il nome di tali istruzioni ha la forma `s`**cc**, dove **cc** è un codice di 2 caratteri che indica la specifica condizione controllata dall'istruzione
- I possibili valori di **cc** e i rispettivi significati sono gli stessi usati dalle istruzioni `b`**cc**
- Le istruzioni `s`**cc** hanno un operando in formato `byte`, che può essere una parola memoria-ordinaria o un registro dati; se la condizione indicata da **cc** risulta vera rispetto all'attuale contenuto di `ccr`, l'operando assume il valore `$FF`, altrimenti assume valore 0

Esempi di Traduzione da *UC* a MC68000-ASM1

- Code1_m68k mostra come tradurre un'istruzione `if` la cui espressione di controllo è un'espressione relazionale su variabili intere `h` e `k`
- Il valore di `h` è contenuto in `d1`, in formato `long`
- Il valore di `k` è contenuto in `d2`, in formato `long`
- Si presti attenzione all'ordine dei registri in `cmp`

Code1_uc

```
if ( h > k ) goto label1;
```

Code1_m68k

```
cmp.l   d2,d1
bgt     label1
```

Esempi di Traduzione da *UC* a MC68000-ASM1

- Le istruzioni di salto non modificano `ccr`; ciò può essere sfruttato se una stessa condizione viene utilizzata per prendere più decisioni, come mostrato dalla traduzione di Code2_uc
- Il valore di `h` è contenuto in `d1`, in formato `word`
- Il valore di `k` è contenuto in `d2`, in formato `word`

Code2_uc

```
if ( h < k ) goto caso_A;
if ( h == k ) goto caso_B;
caso_C:
```

Code2_m68k

```
 cmp.w   d2,d1
 blt     caso_A
 beq     caso_B
caso_C:
```

Esempi di Traduzione da *UC* a MC68000-ASM1

- Quando una variabile viene confrontata con il valore 0, si può usare l'istruzione `tst` che è più efficiente di `cmp`
- Il valore di `h` è contenuto in `d0`, in formato `byte`

Code3_uc

```
if ( w <= 0 ) goto label3;
```

Code3_m68k

```
tst.b   d0
ble     label3
```

Esempi di Traduzione da *UC* a MC68000-ASM1

- Code4_m68k mostra come tradurre in modo efficiente un'istruzione `if` la cui espressione di controllo confronta con 0 il risultato di un calcolo aritmetico, utilizzando il fatto che le istruzioni aritmetiche di MC68000 fanno automaticamente tale confronto
- Il valore di `v1` è contenuto in `d3`, in formato `long`
- Il valore di `v2` è contenuto in `d4`, in formato `long`

Code4_uc

```
v2 += v1;
if ( v2 != 0 ) goto label4;
```

Code4_m68k

```
add.l   d3,d4
bne     label4
```

Esempi di Traduzione da *UC* a MC68000-ASM1

- Code5_m68k mostra come tradurre in modo efficiente un'istruzione `if` la cui espressione di controllo confronta con 0 un valore che è stato copiato, utilizzando il fatto che le istruzioni di trasferimento di MC68000 fanno automaticamente tale confronto
- Il valore di `w1` è contenuto in `d5`, in formato `byte`
- Il valore di `w2` è contenuto in `d6`, in formato `byte`

Code5_uc

```
w1 = w2;
if ( w1 == 0 ) goto label5;
```

Code5_m68k

```
move.b d6,d5
beq    label5
```

Esempi di Traduzione da *UC* a MC68000-ASM1

- Code6_m68k mostra come usare un'istruzione **scc** per memorizzare il risultato di un'espressione relazionale
- Il valore di `x` è contenuto in `d2`, in formato `word`
- Il valore di `y` è contenuto in `d3`, in formato `word`
- Il valore di `z` è contenuto in `d0`, in formato `byte`

Code6_uc

```
z = x <= y;
x += 20;
if ( z != 0 ) goto label6;
```

Code6_m68k

```
cmp.w    d3,d2
sle      d0
add.w    #20,d2
tst.b    d0
bne      label6
```

Condizioni in MIPS32

- In merito al prendere decisioni sulla base di condizioni, gli *ASM-PM* MIPS, sono tra i pochi a non adottare il tipico approccio dei linguaggi *ASM*
- Al contrario, adottano un approccio simile a quello degli *HLL*: la determinazione di una condizione e la decisione su come proseguire la computazione in base a tale condizione, vengono effettuate da una singola, indivisibile, istruzione
- MIPS32 ha istruzioni di salto condizionato e istruzioni di memorizzazione di condizione che determinano una condizione e usano tale valore per produrre un effetto immediato
- Le condizioni non vengono memorizzate in registri di stato, diversamente da quanto accade nella maggior parte degli *ASM-PM*

Istruzioni di Salto

- In MIPS32-MARS, le definizioni di label adottano la sintassi e la semantica usate da quasi tutti gli *ASM*, descritte in precedenza
- Le istruzioni di salto incondizionato sono `b`, `j` e `jr`;
  - `b` e `j` hanno destinazione statica
  - `jr` ha destinazione dinamica, la approfondiremo in future lezioni
- Sia `b` che `j` hanno un solo operando, che deve essere una label
- L'istruzione di destinazione del salto è l'istruzione che segue la definizione della label

Istruzioni di Salto

- In merito alle istruzioni di salto condizionato, vi sono delle differenze tra versioni diverse della famiglia MIPS: vi sono istruzioni presenti in alcune versioni ma non in altre
- In LPS descriveremo e utilizzeremo esclusivamente le istruzioni di MIPS32-MARS
- Le istruzioni di salto condizionato in MIPS32-MARS dispongono o di 2 o di 3 operandi

Istruzioni di Salto

- Nelle istruzioni di salto condizionato a 2 operandi, il primo operando è un registro e il secondo una label
- Tali istruzioni confrontano il contenuto del registro con il valore 0 e, se la specifica condizione controllata dall'istruzione risulta vera, saltano all'istruzione che segue la definizione della label
- Le istruzioni di salto condizionato a 2 operandi hanno la forma b**cc**z, dove **cc** è un codice di 2 caratteri che indica la specifica condizione controllata dall'istruzione

Istruzioni di Salto

- Nelle istruzioni di salto condizionato a 3 operandi, il primo operando è un registro, il secondo un registro o un operando immediato e il terzo una label
- Le istruzioni confrontano il valore dei primi due operandi e, se la specifica condizione determinata dall'istruzione risulta vera, saltano all'istruzione che segue la definizione della label
- In questa presentazione, introduciamo le istruzioni che hanno la forma b**cc**, dove **cc** è un codice di 2 caratteri che indica la specifica condizione determinata dall'istruzione

Istruzioni di Salto

- I possibili valori del codice **cc**, sia per le istruzioni b**cc**z, sia per le istruzioni b**cc**, sono riportati nella tabella seguente

Codice	Condizione	Operatore C corrispondente
`eq`	*Equal*	==
`ne`	*Not Equal*	!=
`lt`	*Less Than*	<
`le`	*Less Than or Equal*	<=
`gt`	*Greater Than*	>
`ge`	*Greater Than or Equal*	>=

- Ulteriori informazioni in una prossima lezione e in **[MIPS32]**

Istruzioni di Memorizzazione di Condizione

- In MIPS32-MARS sono disponibili istruzioni che memorizzano in un registro la condizione relativa al sussistere di una determinata relazione d'ordine tra i valori di due operandi
- Sono istruzioni a 3 operandi: il primo e il secondo sono registri, il terzo può essere un registro o una parola memoria-immediata
- Le istruzioni confrontano il valore del secondo e del terzo operando e memorizzano nel primo operando
  - il valore 1 se la specifica condizione determinata dall'istruzione risulta vera
  - il valore 0 altrimenti

Istruzioni di Memorizzazione di Condizione

- La specifica condizione determinata da ciascuna istruzione è indicata dal codice di 2 caratteri **cc**, contenuto nel nome dell'istruzione
- I possibili valori di **cc** e i rispettivi significati sono gli stessi usati dalle istruzioni b**cc**
- Il nome delle istruzioni di memorizzazione di condizione ha la forma s**cc**, tranne che per il codice `lt`, in corrispondenza del quale MIPS32-MARS presenta una irregolarità sintattica
- Il nome dell'istruzione che memorizza la condizione indicata dal codice `lt`, è
  - `slt` se il terzo operando è un registro
  - `slti` se il terzo operando è una parola memoria-immediata

Esempi di Traduzione da *UC* a MIPS32-MARS

- Code7_mips mostra come tradurre in modo efficiente un'istruzione `if` la cui espressione di controllo confronta una variabile con il valore 0
- Il valore di `x` è contenuto in `t2`

Code7_uc

```
if ( x <= 0 ) goto label7;
```

Code7_mips

```
blez    $t2,label7
```

Esempi di Traduzione da *UC* a MIPS32-MARS

- Code8_mips è un altro esempio di traduzione di un'istruzione `if` la cui espressione di controllo confronta una variabile con il valore 0
- Il valore di `z` è contenuto in `t4`

Code8_uc

```
if ( z != 0 ) goto label8;
```

Code8_mips

```
bnez    $t4,label8
```

Esempi di Traduzione da *UC* a MIPS32-MARS

- Code9_mips utilizza un'istruzione di salto condizionato a 3 operandi, con un registro come secondo operando, per tradurre un'istruzione `if` la cui espressione di controllo confronta due variabili
- Il valore di `b` è contenuto in `s1`
- Il valore di `c` è contenuto in `s2`
- Si presti attenzione all'ordine dei registri nell'istruzione di salto condizionato: è opposto rispetto a quello nell'istruzione `cmp` di MC68000-ASM1

Code9_uc

```
if ( b <= c ) goto label9;
```

Code9_mips

```
ble     $s1,$s2,label9
```

Esempi di Traduzione da *UC* a MIPS32-MARS

- Code10_mips utilizza un'istruzione di salto condizionato a 3 operandi, con una parola memoria-immediata come secondo operando, per tradurre un'istruzione `if` la cui espressione di controllo confronta una variabile con una costante diversa da 0
- Il valore di `d` è contenuto in `s3`

Code10_uc

```
if ( d > 100 ) goto label;
```

Code10_mips

```
bgt     $s3,100,label
```

Esempi di Traduzione da *UC* a MIPS32-MARS

- Code11_mips utilizza un'istruzione di memorizzazione di condizione, per tradurre un'istruzione che assegna ad una variabile il risultato di un'espressione relazionale
- I valori di `r`, `x`, `y` sono contenuti, rispettivamente, in `s0`, `t1`, `t2`

Code11_uc

```
r = x >= y;
```

Code11_mips

```
sge      $s0,$t1,$t2
```

Sessione di Esercizi: Controllo_Esecuzione_ASM

- Svolgere il gruppo di esercizi **Controllo_Esecuzione_ASM** su *Edu99*
- Per ulteriori spiegazioni sui contenuti degli esercizi, si vedano
  - esempi *Execution Control* su *Edu99*
  - **[M68000]**
  - **[MIPS32]**