Esercizio 1 (11 punti).

Sia data una CPU con processore a **8GHz** e **8 CPI** (Clock per Instruction) che adoperi indirizzi da 32 bit e memoria strutturata su due livelli di **cache** (L1, L2), il cui setup è come segue:
L1 è una cache **set-associativa** a **3 vie** con **2 set** e **blocchi da 4 word**; adopera una politica di rimpiazzo **LRU**. Ricordiamo che consideriamo il set come l'insieme del blocco in cache con tag e bit di validità, mentre la linea è il gruppo di set con il medesimo indice.
L2 è una cache **direct-mapped** con **8 linee** e **blocchi da 32 word**.

1) Supponendo che all'inizio nessuno dei dati sia in cache, indicare quali degli accessi in memoria indicati di seguito sono HIT o MISS in ciascuna delle due cache. Per ciascuna **MISS** indicare se sia di tipo Cold Start (**Cold**), Capacità (**Cap**) o Conflitto (**Conf**). Utilizzare la tabella sottostante per fornire i risultati ed indicare la metodologia di calcolo più in basso.

	Address	8076	8080	4096	4100	2000	2040	2076	2096	8080	8084	4104	4196
L1	**Block#**												
	Index												
	Tag												
	HIT/MISS												
	Miss type												
L2	**Block#**												
	Index												
	Tag												
	HIT/MISS												
	Miss type												

2) Calcolare le dimensioni in bit (compresi i bit di controllo ed assumendo che ne basti uno per la LRU) delle due cache: (a) L1 e (b) L2.

3) Assumendo che gli accessi in **memoria** impieghino **200 ns**, che gli **hit** nella cache **L1** impieghino **2 ns** e gli **hit** nella cache **L2** impieghino **50 ns**, calcolare (a) il **tempo totale** per la sequenza di accessi, (b) il tempo **medio** per la sequenza di accessi, e (c) **quante istruzioni** vengono svolte nel tempo medio calcolato.

4) Calcolare il word offset del sesto indirizzo (2040) per la cache L2 spiegando i calcoli effettuati.

5) Supponendo che gli indirizzi nella tabella siano virtuali e la memoria virtuale consti di **512 pagine** di **4KiB** ciascuna, indicarne i numeri di pagina virtuale. Si assume che la cache sia a monte della memoria virtuale.

Address	8076	8080	4096	4100	2000	2040	2076	2096	8080	8084	4104	4196
Page#												

Esercizio 2 (11 punti).

Considerare l'architettura MIPS a ciclo singolo nella figura in basso (e in allegato).
Si vuole aggiungere alla CPU l'istruzione di salto condizionato **branch on less than to address in memory** (`bltam`), di **tipo I** e sintassi assembly
`bltam` **`$op1, $op2, where`**
(dove op1, op2 e `where` sono rispettivamente i campi `rs`, `rt` e il campo immediate dell'istruzione) che:
1) calcola se op1 sia strettamente minore di op2 (`op1 < op2`)
2) se `$op1 < $op2` allora opera come segue:
    se `where` è un indirizzo di word valido, ossia multiplo di 4, procede come segue:
        carica dalla memoria una word all'indirizzo che ha i 16 MSB pari a `0x1001` e i 16 LSB pari a `where`;
        salva nel registro `$at` l'indirizzo dell'istruzione seguente a `bltam`;
  se `$op1 ⩾ $op2` oppure `where` *non* è un indirizzo di word valido, non effettua alcun salto e non salva alcun valore in `$at`.

Esempio: Supponiamo che `$op1` sia `$s0`, contenente `0x0000 000C`, che `$op2` sia `$s1`, contenente `0x1001 1010`,
che `where` valga `0x0100`. e che l'istruzione corrente sia all'indirizzo `0x0040 0080`
In tal caso, abbiamo che `$op1 < $op2` e `where` è un indirizzo valido per word in memoria.
Dunque, da `where` si ricava l'índirizzo `0x1001 0100` usando il prefisso `0x1001` come al punto (2) e si carica la word da memoria a quell'indirizzo.
Supponiamo che all'indirizzo `0x1001 0100` in memoria ci sia questa word: `0x0040 10D8`.
Dunque si effettua il salto all'istruzione `0x0040 10D8`. Attenzione: *non* a `0x1001 0100` ma a `0x0040 10D8`!
Inoltre, viene salvato l'indirizzo successivo a quello dell'istruzione corrente nel registro `$at`, ossia `$at`=`0x0040 0084`.
Si noti che se si fosse verificato che `$op1 ⩾ $op2` o se `where` non fosse stato un indirizzo di word valido, non si sarebbe svolta alcuna delle operazioni menzionate.

1) Mostrare le **modifiche all'architettura** della CPU MIPS, avendo cura di aggiungere eventuali altri componenti necessari a realizzare l'istruzione. A tal fine, si può alterare la stampa del diagramma architetturale oppure ridisegnare i componenti interessati dalla modifica, avendo cura di indicare i fili di collegamento e tutti i segnali entranti ed uscenti. Indicare inoltre sul diagramma i **segnali di controllo** che la CU genera *per realizzare l'istruzione*.

2) Indicare il contenuto in bit della word che esprime l'istruzione
`bltam` **`$s4, $s6, 0x0100`**
compilando la tabella sottostante (assumiamo che lo *OpCode* di `bltam` sia `0x3D`)

3) Supponendo che l'accesso alle **memorie** impieghi **200 ns**, l'accesso ai **registri 50 ns**, le operazioni dell'**ALU e** dei **sommatori 100 ns**, e che gli altri ritardi di propagazione dei segnali siano trascurabili, indicare la durata totale del **ciclo di clock** tale che anche l'esecuzione della nuova istruzione sia permessa *spiegando i calcoli effettuati*.

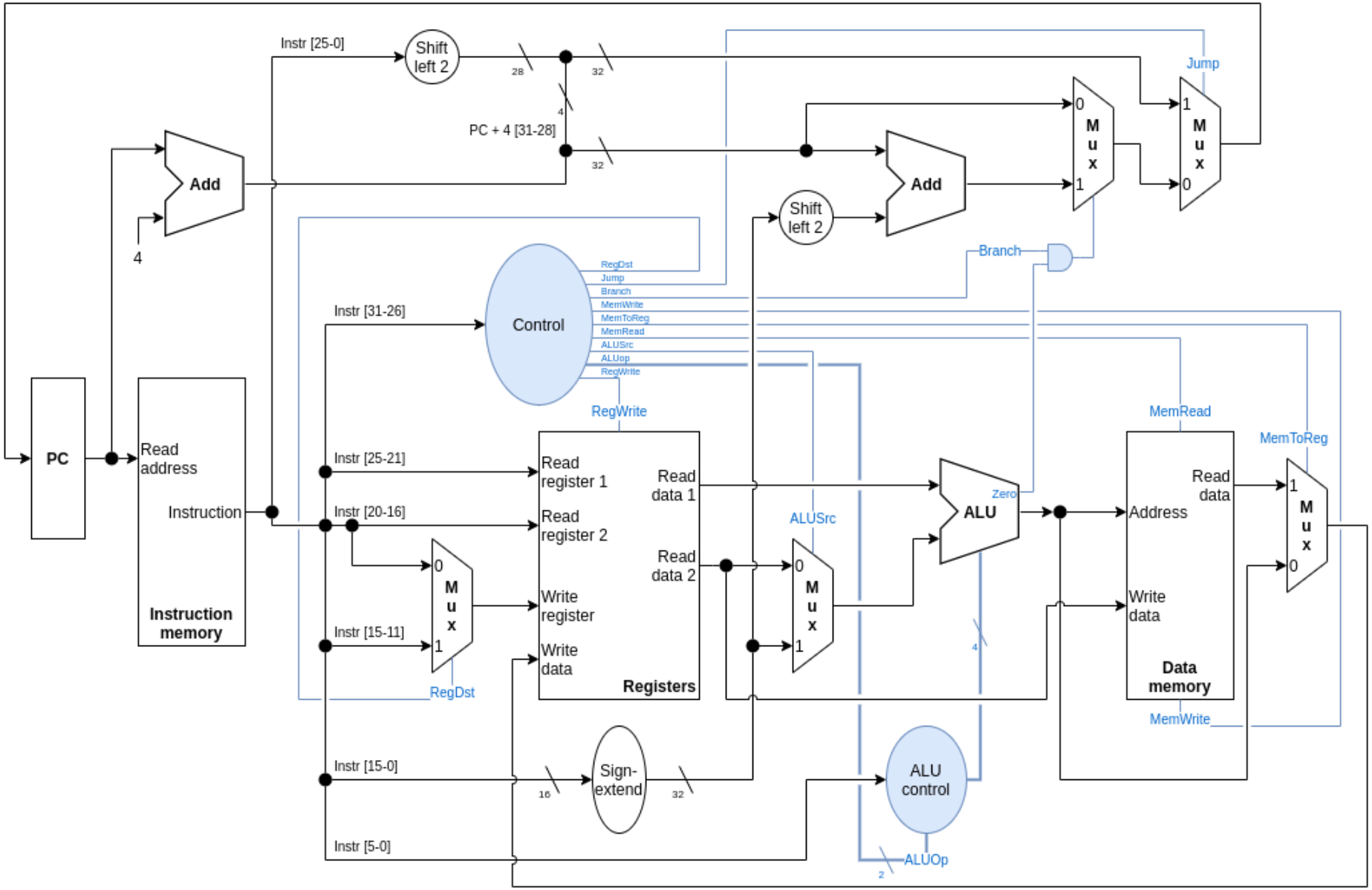
4) Indicando con `Bltam` il segnale di controllo che viene asserito per eseguire la nuova istruzione, assumiamo che
  a) tutti i segnali di tipo *don't care* siano pari a `0` e che

b) la Control Unit della CPU MIPS modificata per supportare `Bltam` sia difettosa e sovrascriva il segnale `Branch` come segue:

`Branch` ≔ `Jump` (il simbolo ≔ denota che la variabile a sinistra assume il valore dato della variabile a destra)

In tal caso, indicare quale valore sia assegnato a `$v0` al termine dell'esecuzione del seguente frammento di codice, assumendo che i registri `$v0`, `$s0` e `$s1` siano inizializzati a `0`. Motivare la propria risposta.

```
      addi $s0,  $0, 14
      add  $s1, $s0, $0
      beq  $s1, $s0, Out
      j    Exit
Out:  subi $s0, $s0, 2
Exit: addi $v0, $s0, 2
```


Instr [25-0]
Shift left 2
28
32
4
PC + 4 [31-28]
32
Add
4
Add
Shift left 2
0
M
u
x
1
1
M
u
x
0
Jump
Branch
RegDst
Jump
Branch
MemWrite
MemToReg
MemRead
ALUSrc
ALUop
RegWrite
Control
Instr [31-26]
RegWrite
PC
Read address
Instruction
Instruction memory
Instr [25-21]
Instr [20-16]
Instr [15-11]
Instr [15-0]
Instr [5-0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
RegDst
ALUSrc
Zero
ALU
4
MemRead
MemToReg
Read data
Address
Write data
Data memory
MemWrite
Sign-extend
16
32
ALU control
ALUOp
2

Esercizio 3 (11 punti)

Si consideri l'architettura MIPS con pipeline mostrata nella figura in basso (e in allegato). Il programma qui di seguito somma i primi L numeri nella sequenza fornita, limitando il valore massimo di ogni elemento a 4 (un elemento maggiore di 4 verrà limitato al valore 4). La stringa finale viene stampata se la somma totale è maggiore o uguale a 12.

```
1  .data
2  A: .byte 5, 4, 2, 1, 0, 0, 0, 0
3  L: .byte 5
4  S: .asciiz "\nThe result is bigger or equal than 12!"
5  # Risultato atteso: 11
6  .text
7  main: la   $a0, A            # Indirizzo base array input
8        and  $v0, $zero, $0    # $v0: risultato
9        lb   $a2, L            # Lunghezza dell'array (può essere minore di A)
10       li   $t2, 4            # intero salvato su $t2
11       sub  $a1, $v0, $v0     # $a1: Indice corrente
12 cycl: beq  $a1, $a2, exit    # Array scorso? Esci
13       add  $t1, $a0, $a1     # $t1: Indirizzo corrente
14       lb   $s1, ($t1)        # $s1: Elemento corrente
15       slt  $at, $t2, $s1     # $at vale 1 se $s1 > 4
16       beq  $at, $zero, sum   # Se $s1 <= 4, somma
17       li $s1, 4              # Altrimenti, clip to 4
18 sum:  add  $v0, $v0, $s1     # Somma $s1
19       addi $a1, $a1, 1       # Passa a cella successiva
20       j    cycl              # Riprendi il ciclo
21 exit: add  $s0, $v0, $0      # Salva risultato in $s0
22       addi $v0, $0, 1        # Imposta stampa interi
23       add  $a0, $s0, $0      # Imposta stampa ris.
24       syscall                # Stampa
25       blt  $s0, 12, term     # Se $s0 >= 12, stampa
26       la   $a0, S            # Carica la stringa S
27       addi $v0, $zero, 4     # Imposta stampa
28       syscall                # Stampa
29 term: addi $v0, $0, 10       # Imposta uscita
30       syscall                # Esci
```

Si supponga che *tutte le istruzioni impiegate facciano parte del set supportato dalla CPU in figura*, ossia non si fa uso di alcuna pseudoistruzione.

Si indichino (ignorando hazard che possano concernere la syscall):

1) le istruzioni tra le quali sono presenti **data hazard** – indicare i numeri di istruzione N ed M (visibili alla sinistra delle linee di codice in figura) ed il registro coinvolto $xY (nel formato N / M / $xY);

2) le istruzioni tra le quali sono presenti **control hazard** – indicare i numeri di istruzione N ed M (nel formato N / M);

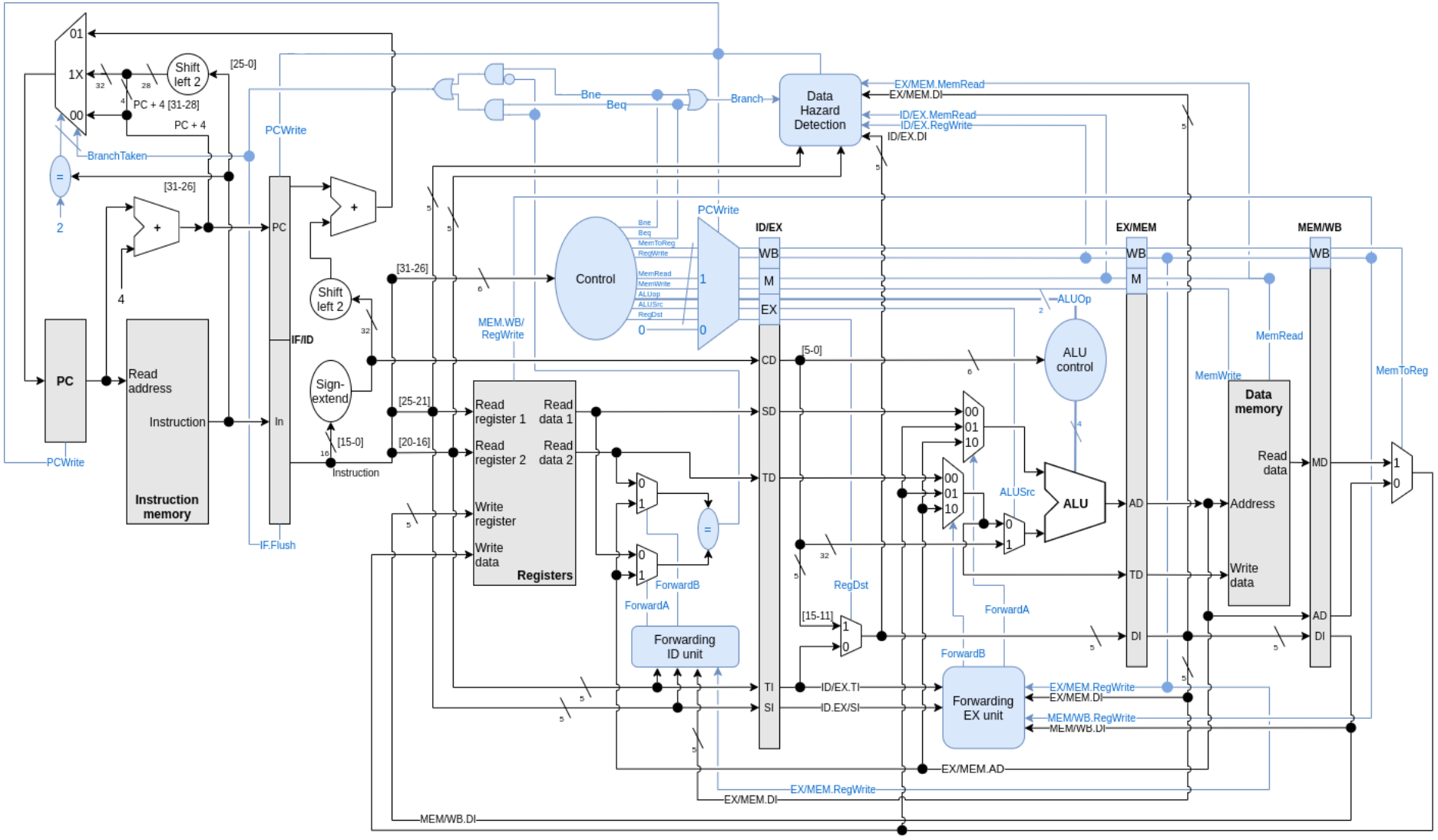
3) quanti **cicli di clock** sono necessari ad eseguire il programma tramite **forwarding**, spiegando il calcolo effettuato;

4) quanti **cicli di clock** sarebbero necessari ad eseguire il programma **senza forwarding**, spiegando il calcolo effettuato;

5) quali sono, per ognuna delle cinque fasi, le **istruzioni** (o le bolle) in pipeline durante il **14° ciclo di clock** (con **forwarding**);

IF:
ID:
EX:
MEM:
WB:

6) qual è il ruolo delle connessioni al multiplexer a monte del PC nel diagramma seguente, spiegando nel maggior dettaglio possibile i tre possibili input e il ruolo dei due bit di controllo.


01
1X
00
Shift left 2
[25-0]
32
28
4
PC + 4 [31-28]
PC + 4
PCWrite
BranchTaken
=
[31-26]
2
+
PC
IF/ID
In
Shift left 2
Sign-extend
[15-0]
16
Instruction
IF.Flush
PC
PCWrite
Read address
Instruction
Instruction memory
Bne
Beq
Branch
Data Hazard Detection
EX/MEM.MemRead
EX/MEM.DI
ID/EX.MemRead
ID/EX.RegWrite
ID/EX.DI
Control
Bne
Beq
MemToReg
RegWrite
MemRead
MemWrite
ALUop
ALUSrc
RegDst
PCWrite
ID/EX
WB
M
EX
CD
SD
TD
TI
SI
MEM.WB/ RegWrite
[31-26]
[25-21]
[20-16]
[5-0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
ForwardB
ForwardA
Forwarding ID unit
ALU control
ALUOp
ALUSrc
ALU
RegDst
[15-11]
ID/EX.TI
ID.EX/SI
Forwarding EX unit
EX/MEM.RegWrite
EX/MEM.DI
MEM/WB.RegWrite
MEM/WB.DI
EX/MEM.AD
EX/MEM
AD
TD
DI
MemRead
MemWrite
Data memory
Read data
Address
Write data
MEM/WB
MD
MemToReg
MEM/WB.DI