CHALMERS

Dec 7 2009

DAT105 – COMPUTER ARCHITECTURE.

EXAM SOLUTIONS FOR 2009-12-17

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Chalmers University of Technology
SE 412 96 Göteborg, Sweden

Phone: +46 (0)31-772 10 00
E-mail: labe@chalmers.se
Web: www.ce.chalmers.se/~labe

<u>Assignment 1.</u>

A)

(i) $T_A = (1+2+4)/3 = 2.3$; $\mathbf{T_B = (2+2+2)/3 = 2}$; $T_R = (6+1+1)/3 = 2.7$

B have the highest performance when using arithmetic mean.

Using arithmetic mean to compare programs results in that programs with longer execution times impacts the result higher than programs with shorter execution times.

(ii), (iii) :

	Comp A	**Comp B**	**Ref comp**
P1/R	**1/6**	**2/6**	**1**
P2/R	**2**	**2**	**1**
P3/R	**4**	**2**	**1**
Geometric mean	**1.1**	**1.1**	**1**

A & B have the same performance when using the geometric mean of the normalized execution times.

B.

Amdahl's law : $\text{SpeedUp}_{\text{overall}} = 1 / (1 – F + F/S)$; where F = Fraction of the total time that can be sped up. S = Speedup factor (=no. of processors)

$\text{SpeedUp}_{\text{overall}} = 1 / (1 - 0.8 + 0.8/S) = 1 / (0.2 + 0.8/S)$;

Let S -> infinity: $\text{SpeedUp}_{\text{overall}}$ -> 1 / 0.2 = 5.

C.

$\mathbf{T_{exe} = IC * CPI_{ave} * T_c}$

$CPI_{ave} = CPI_{base} * 0.8 + CPI_{memrefs} * 0.2 = 1*0.8 + 2*0.2 = 1.2$;

$T_{exe} = 10^9 * 1.2 * 10^{-9} = 1.2$ seconds.

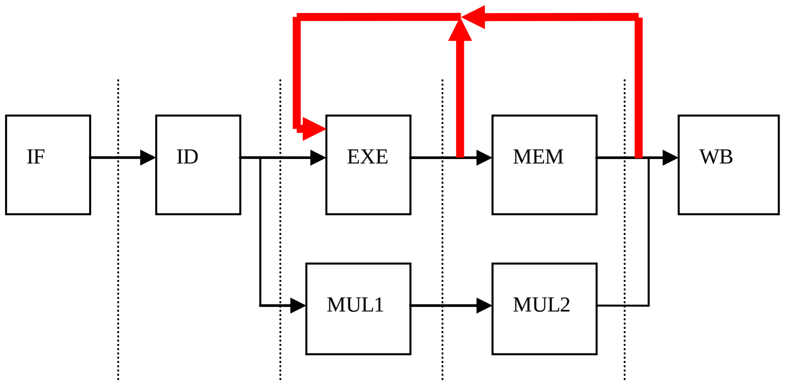
Assignment 2.

A)
(i)

```
LOOP:   LD       R4, 0(R1)
                  ↖↘
        DMUL     R5, R4, R4                RAW dependence on R4[1]
                  ↕
        SD       R5, 0(R1)                 RAW dependence on R5[2]

        DADDI    R1, R1, 8
                  ↕
        BNE      R1, R2, LOOP              RAW dependence on R1[3]
```

Assuming a 5-stage pipeline structure using forwarding/bypass for handling of RAW hazards (shown as red thick arrows):



(ii) See execution diagram in (iii)
1st RAW: Partly resolved via forward MEM -> EXE/MUL1
2nd RAW: Partly resolved via forward MEM/MUL2 -> EXE
3rd RAW: Resolved via forward EXE -> EXE

(iii) Control hazard (branch delay slots) = 2; RAW = 2 stall cycles.

Cycle No.	1	2	3	4	5	6	7	8	9	10	11
LD R4,0(R1)	IF	ID	EXE	MEM	WB						
DMUL R5,R4,R4		IF	ID	stall	MUL1	MUL2	WB				
SD R5,0(R1)			IF	stall	ID	stall	EXE	MEM	WB		
DADDI R1,R1,8					IF	stall	ID	EXE	MEM	WB	
BNE R1,R2,LOOP							IF	ID	EXE	MEM	WB
Delay slot 1								IF	ID	EXE	…..
Delay slot 2									IF	ID	…..

Number of cycles per vector element is 9 cycles.

B) Assume n is an even number. Then unroll the loop once and put “NOP” in the delay slots.

```
LOOP:   LD       R4, 0(R1)
        DMUL     R5, R4, R4
        SD       R5, 0(R1)
        DADDI    R1, R1, 8
        NOP                          delay slot
        NOP                          delay slot

        LD       R4, 0(R1)
        DMUL     R5, R4, R4
        SD       R5, 0(R1)
        DADDI    R1, R1, 8
        BNE      R1, R2, LOOP
        NOP                          delay slot
        NOP                          delay slot
```

Now, move the second LD up, the second SD down, merge the two DADDI’s, adjust offsets, and rename R4 and R5 :

```
LOOP:   LD      R4, 0(R1)
        LD      R3, 8(R1)               R4 renamed to R3

        DMUL    R5, R4, R4
        DMUL    R6, R3, R3              R5 renamed to R6

        DADDI   R1, R1, 16

        BNE     R1, R2, LOOP

        SD      R5, -16(R1)
        SD      R6, -8(R1)              R5 renamed to R6
```

Cycle No.	1	2	3	4	5	6	7	8	9	10	11	12
LD R4,0(R1)	IF	ID	EXE	MEM	WB							
LD R3,8(R1)		IF	ID	EXE	MEM	WB						
DMUL R5,R4,R4			IF	ID	MUL1	MUL2	WB					
DMUL R6,R3,R3				IF	ID	MUL1	MUL2	WB				
DADDI R1,R1,16					IF	ID	EXE	MEM	WB			
BNE R1,R2,LOOP						IF	ID	EXE	MEM	WB		
SD R5,-16(R1)							IF	ID	EXE	MEM	WB	
SD R6,-8(R1)								IF	ID	EXE	MEM	WB

(i) Two times (assuming n is even)

(ii) Two rename registers (R3 and R6)

(iii) New loop has 8 instructions so each loop now has 3 more instructions. However, since two vector elements are now computed in 8 cycles the number of cycles per vector element is reduced to 4 cycles.

C) In the C assignment we do not use loop unrolling but just reschedule the code within one loop iteration. To reduce control hazards we can move the SD instruction down into one of the delay slots. One cycle is then lost due to control hazard :

```
LOOP:   LD      R4, 0(R1)
        DMUL    R5, R4, R4
        DADDI   R1, R1, 8
        BNE     R1, R2, LOOP
        SD      R5, -8(R1)
        NOP                         Delay slot
```

Note. Moving also the DMUL instruction into the delay slots will not help since then we would have an RAW hazard on R5 which is not solved by forwarding.

D) See book pages 82-83 and Figure 2.4. We would have one entry corresponding to the BNE instruction in the end of the program. The prediction would evolve according to this table if we assume we start in state 00 and assuming n > 5:

Execution of BNE instruction	Prediction state before exec	Prediction	State after exec
first	00	Not taken (wrong)	01 (it was taken)
second	01	Not taken (wrong)	11 (it was taken)
third	11	Taken (correct)	11 (it was taken)
4th, 5th, etc	11	Taken (correct)	11 (it was taken)
Iteration n	11	Not taken (wrong)	10 (it was not taken)

Assignment 3.

A)	SUB	F0, F1, F2	
	DIV	F3, F0, F4	WAR name dependence on F4
	ADD	F4, F5, F6	

The three stages of Tomasulo's Algorithm :

1. Issue (I)
- Issue instruction if no structural hazard for a reservation station
- Read already available operands

2. ***Execution (EX)***
- Execute when both operands are available;
  if operad(s) not ready, watch Common Data Bus (CDB) for result

3. ***Write result* (WR)**
- Write on CDB to all awaiting functional units;
- Mark reservation station available.

B)

Cycle #:	**1**	**2**	**3**	**4**	**5**	**....**	**9**	**10**	**11**	**12**	**....**	**19**	**20**
SUB	**I**	**EX**	**EX**	**EX**		**....**	**EX**	**WR**					
DIV		**I**			**....**			**EX**	**EX**	**EX**	**….**	**EX**	**WR**
ADD			**I**	**EX**	**EX**	**….**	**EX**	**WR**					

The last instruction (ADD) write to the register file in cycle #10.

C)

The four steps of a Speculative Tomasulo algorithm:

1. Issue

- If reservation station ***and reorder buffer slot*** free, issue instr & send operands ***& reorder buffer nr. for destination***

2. Execution

- If both operands ready: *execute*; if not, watch CDB for result; when both operands are in reservation station: *execute*

3. Write result

- Write on Common Data Bus to all awaiting FUs ***& reorder buffer***; mark reservation station available

4. Commit* — *update register with reorder result

- When instr. is at head of reorder buffer & result is present; update register with result (or store to memory) and remove instr. from reorder buffer

D) F0 is supplied via the CDB bus (in cycle #10 in the example above).

Assignment 4.

A)

Compulsory (or cold) *miss*: The first reference to a block is always a miss.

Capacity *miss*: If the space is not sufficient to host the data or code that have been accessed.

Conflict *miss*: Two memory blocks may be mapped to the same cache block with a direct or set-associative address mapping even if there is still unused space in cache.

B)

Average memory access time = Hit time + Miss rate x Miss penalty.

C)

Way prediction addresses "Hit time". Used in Set-associative caches to predict in which Set a block resides. If correctly predicted Hit time is reduced because the signal path to read out the block may be preset for that Set (means selecting entries in the multiplexers). If mispredicted typically a one clock cycle miss penalty is experienced. See text book on page 295 and Lecture 4.

D)

Multibanked Caches:

– Divide cache blocks into banks that can be accessed simultaneously

– While one bank is accessed for possibly several cycles, next access can proceed if it goes to another bank

– Bank is selected based on block address

See as an example fig. 5.6 on page 298.

E) Loop interchange: The compiler interchanges loops so variables are accessed according to their storing order in memory (improving locality). The example below is used when the A matrix is stored in Row major order, i.e. row by row.

Before:

```
for (col=0; col < N; col++)
        for (row = 0; row < N; row++)
                A [row, col] = ...
```

After:

```
for (row = 0; row < N; row++)
        for (col = 0; col < N; col++)
                A [row, col] = ...
```

See lecture 4 slides p. 40.

Assignment 5.

A) Fine-Grain: switch thread on each clock cycle.
Coarse-Grain: switch thread on long latency operation (e.g. cache misses)

B) Instruction Fetch front-ends added per thread and register file structures. May also involve adding more associativety and sizes of cache and TLB structures. And number and size of load/store queues. See example in text book p. 176-177.

C) MIMD = Multiple Instruction Streams Multiple Data Streams. Eaxh processor has its own instruction flow operated on a unique flow om data.

D) All processors need to monitor the addresses sent on the bus. If a write occurs all other processors marks this block as invalid in their cache. On cache misses, a processors holding a dirty copy of the block supplies that block in response to the read request and causes the memory access to be aborted. See textbook on page 208-.