King Fahd University of Petroleum and Minerals
College of Computer Sciences and Engineering
Department of Computer Engineering

COE 308 – Computer Architecture (T041)

Homework # 03 (*due date: Sunday 28/11/2004*)

***** Show all your work. No credit will be given if work is not shown! *****

Problem # 1 (30 points): A system has both a virtual memory and a real address data cache, with the following characteristics:

64-bit virtual address
32-bit real address
4 Kbyte pages
4-way set associative TLB with 256 entries in total
Direct mapping data cache with 512 blocks in total
Each block consists of 8 words

(i) **(20 points)** Show the division of bits in the real address (i.e. Cache TAG, SET, and WORD sizes), and in the virtual address (i.e. Page, Line, TLB TAG, TLB INDEX, SET, and WORD sizes).
(ii) **(10 points)** How many virtual pages are there in the system? How many real pages are there in the system?

Solution:

(i) Virtual pages = 2^{64} / 4K = 2^{52} pages ⇒ "Page" bits = 52 bits
Page size = 4K = 2^{12} ⇒ "Line" bits = 12 bits
TLB sets = 256 / 4 = 64 ⇒ TLB INDEX bits = $\log_2 64$ = 6 bits
Since virtual pages = 2^{52} pages that can be mapped to 64 TLB sets ⇒ TLB TAG bits = $\log_2 (2^{52} / 64)$ = 46 bits
WORD size = 8 ⇒ Page/Cache WORD bits = $\log_2 (8)$ = 3 bits
Cache SETs = 512 ⇒ Cache SET bits = $\log_2 (512)$ = 9 bits
Cache TAGs = MM blocks / Cache SETs = $(2^{32} / 8) / 512 = 2^{20}$ ⇒ Cache TAG bits = 20 bits

Notes:
- "Page" bits + "Line" bits = 52 + 12 = 64 bits = virtual address size
- TLB INDEX bits + TLB TAG bits = 6 + 46 = 52 bits = "Page" bits
- Cache WORD bits + Cache SET bits + Cache TAG bits = 3 + 9 + 20 = 32 bits = real address size
- Cache WORD bits + Cache SET bits = 3 + 9 = 12 bits = "Line" bits ⇒ All Cache TAG bits are coming from TLB (nothing coming from virtual address)

(ii) Virtual pages = 2^{64} / 4K = 2^{52} pages
Real pages = 2^{32} / 4K = 2^{20} pages

Problem # 2 (30 points): Assume a pipelined processor with 5 stages: Instruction Fetch (IF), Instruction Decode (ID), Operand Fetch (OF), Instruction Execute (IE), and Operand Store (OS). Consider the following code:

```
            LD      R1, 3           ; R1 ← 3
            LD      R2, 2           ; R2 ← 2
            LD      R3, –1          ; R3 ← –1
            ADD     R2, R2, R3      ; R2 ← R2 + R3
Repeat:     SUB     R1, R1, R2      ; R1 ← R1 – R2
            BZ      Skip            ; Branch to "Skip" if result is Zero
            SUB     R2, R1, R2      ; R2 ← R1 – R2
            SUB     R2, R2, R3      ; R2 ← R2 – R3
            BNN     Repeat          ; Branch to "Repeat" if result is Not Negative
Skip:       ADD     R3, R2, R3      ; R3 ← R2 + R3
```

(i) **(10 points)** Calculate the number of cycles to execute the code using a non-pipelined processor assuming that each instruction requires 5 cycles to execute. Show the contents of R1, R2, and R3 during the execution.

Instruction	R1	R2	R3
1	3		
2	3	2	
3	3	2	-1
4	3	1	-1
5	2	1	-1
6	Branch ***not*** taken		
7	2	1	-1
8	2	2	-1
9	Branch taken		
5	0	2	-1
6	Branch taken		
10	0	2	1

cycles to execute = 12 × 5 = 60 cycles

(ii) **(15 points)** Use the following table to calculate the number of cycles to execute the code using the pipelined processor described above taking into account **data dependency**. Assume no forwarding hardware. Assume also that when a branch instruction is **fetched**, the pipeline is stalled until target address is calculated and branch decision is made. Assume delayed branch instruction scheme **not** being used.

OS					1	2	3			4			5	6				7			8	9				5	6				10
IE				1	2	3			4			5	6				7			8	9				5	6				10	
OF			1	2	3	***4***	***4,5***	4	***5,6***	***5,6***	5	6				7	***8***	***8,9***	8	9				5	6				10		
ID		1	2	3	4	5	6								7	8	9						5	6				10			
IF	1	2	3	4	5	6								7	8	9						5	6				10				
Time unit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

(iii) **(5 points)** Calculate the speedup.

Speedup = (# cycles to execute using non-pipelined processor) / (# cycles to execute using pipelined processor)
= 60 / 31 = 1.935