Universidad Carlos III de Madrid

Computer Science and Engineering Department
Computer Engineering

Computer Architecture
October 20th 2014

ATTENTION:

- Please read carefully all the questions before starting.
- Books, notes and calculators are not allowed.
- Mobile phones must be disconnected during the test (disconnected, not muted).
- Only answers in ink will be corrected. Please, do not use pencil.
- **Time limit is 90 minutes.**
- **Give every exercise in separate sheets. If you do not answer an exercise, please give a blank sheet with the exercise number.**

NAME:
FAMILY NAME:
NIA:

Exercise 1 [4 points]: Given the following code fragment:

```
bucle:  lw $f0, 0($r1)
        lw $f2, 0($r2)
        mul.f $f4, $f0, $f2
        add.d $f6, $f6, $f4
        addi $r1, $r1, 4
        addi $r2, $r2, 4
        sub $r3, $r3, 1
        bnez $r3, bucle
```

1.1 [0.5 points]: Make a list of all the possible data dependencies, without considering a specific structure for the pipelined architecture. For every dependency you must provide the affected register, source instruction, target instruction and type of dependency.

1.2 [1.5 points]: Make a time diagram (chronogram) for the case of a MIPS architecture with a 5 stage pipeline, with the following constraints:

- No *forwarding* hardware.
- The architecture allows that an instruction writes a register and another instruction reads that register without problem in the same clock cycle.
- Branches are handled by flushing the pipeline.
- Memory references require one clock cycle.
- Branch effective address is computed in the execution stage.

1.3 [0.5 points]: Determine how many cycles are needed to run N iterations of the loop.

1.4 [1 point]: Make a time diagram or chronogram for the MIPS architecture with a 5-stage pipeline with the following constraints:

- There is *forwarding hardware*.
- Assume that branches are handled predicting all branches as taken.

1.5 [0.5 points]: Determine how many cycles are needed to execute N iterations of the loop with the conditions of question 1.4.

SOLUCIÓN:

1.1 If instructions are numbered in the following way:

```
loop:   lw $f0, 0($r1)          #I1
        lw $f2, 0($r2)          #I2
        mul.f $f4, $f0, $f2     #I3
        add.d $f6, $f6, $f4     #I4
        addi $r1, $r1, 4        #I5
        addi $r2, $r2, 4        #I6
        sub $r3, $r3, 1         #I7
        bnez $r3, bucle         #I8
```

A minimum set of dependencies would be:

- $f0: I1 -> I3 (RAW)
- $f2: I2 -> I3 (RAW)
- $f4: I3 -> I4 (RAW)
- $r3: I7 -> I8 (RAW)

1.2 As there is no forwarding, when there is a RAW dependency, WB stage from source instruction must be waited to start ID of the target instruction.

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13
I1: lw $f0, 0($r1)	IF	ID	EX	M	WB								
I2: lw $f2, 0($r2)		IF	ID	EX	M	WB							
I3: mul.f $f4, $f0, $f2			IF	ST	ST	ID	EX	M	WB				
I4: add.d $f6, $f6, $f4						IF	ST	ST	ID	EX	M	WB	
I5: addi $r1, $r1, 4									IF	ID	EX	M	WB
I6: addi $r2, $r2, 4										IF	ID	EX	M
I7: sub $r3, $r3, 1											IF	ID	EX
I8: bnez $r3, bucle												IF	ST

Instruction	14	15	16	17	18	19	20	21					
I6: addi $r2, $r2, 4	WB												
I7: sub $r3, $r3, 1	M	WB											
I8: bnez $r3, bucle	ST	ID	EX	M	WB								
I9: (sig a bnez)		IF	-										
I1: lw $f0, 0($r1)				IF	ID	EX	M	WB					

- Instruction I3, has a stall until I2 has written the read value in $f2. Register file read can be performed in the same cycle when I2 writes (cycle 6).
- Instruction I4 cannot start fetch until fetch unit is not released (cycle 6).
- Instruction I4 has a stall until I3 has written the computed value in $f4. Register file read can be performed in the same cycle when I3 writes in the register file (cycle 9)
- Instruction I5 cannot start fetch until fetch unit is not released (cycle 9).
- Instruction I8 has a stall until I7 has written the value $r3. Register file read can be performed in the same cycle when I7 writes (cycle 15).
- Instruction I9 (next to bnez) cannot start fetch until fetch unit is not released (cycle 15).
- Although branch address is known at the end of ID stage from I8, the branch way (taken/not-taken) is not known until execution stage. Thus fetch is repeated in cycle 17.

1.3: To determine the number of cycles, we need to compute how many cycles are required for any iteration and for the last iteration.

The cost of an iteration different from the last one is obtained determining the number of cycles from the start of I1 until it starts again. These are 16 cycles.

The cost of the last iteration is obtained determining the number of cycles until instruction I8 is finished. Those are 18 cycles.

Cost = 16*n + 2.

1.4: Now, forwarding is allowed when it is posible and no wait is needed until WB.

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13
I1: lw $f0, 0($r1)	IF	ID	EX	M	WB								
I2: lw $f2, 0($r2)		IF	ID	EX	M	WB							
I3: mul.f $f4, $f0, $f2			IF	ID	ST	EX	M	WB					
I4: add.d $f6, $f6, $f4				IF	ST	ID	EX	M	WB				
I5: addi $r1, $r1, 4						IF	ID	EX	M	WB			
I6: addi $r2, $r2, 4							IF	ID	EX	M	WB		
I7: sub $r3, $r3, 1								IF	ID	EX	M	WB	
I8: bnez $r3, bucle									IF	ST	ID	EX	M
I1: lw $f0, 0($r1)											IF	ID	EX

Instrucción	14	15	16	17	18	19	20	21					
I8: bnez $r3, bucle	WB												
I1: lw $f0, 0($r1)	M	WB											

- Instruction I3 may start execution after memory stage from I2 (cycle 6) thanks to forwarding.
- Instruction I4 cannot start decode until decode unit has not been released (cycle 6).
- Instruction I$ can start execution after execution stage of I3 (cycle 7) thanks to forwarding.

- Instruction I5 cannot start fetch until fetch unit is not released (cycle 6).
- Instruction I8 cannot start decode until $r3 value has not been computed (cycle 10) and is passed trhough forwarding (cycle 11).

1.5: Cost of an iteration different from the last one is 10 cycles. Last iteration requires 14 cycles.

Coste = 10 * n + 4.

Exercise 2 [3 points]: Given the following global variable definition:

```
const unsigned int max = 1024 * 1024;
double x[max];
double y[max];
double z[max];
double vx[max];
double vy[max];
double vz[max];
```

And the following function:

```
void update_positions(double dt) {
  for (unsinged int i=0; i<max; ++i) {
    x[i] = vx[i] * dt + x[i];
    y[i] = vy[i] * dt + y[i];
    z[i] = vz[i] * dt + z[i];
  }
}
```

Assume that the system has a L1 4-way associative cache with size of 32 KB and a line size of 64 bytes. L2 cache is 8-way associative with a size of 1 MB and a line size of 64 bytes. Replacement policy is LRU.

The arrays are stored consecutively in memory and the first of them is stored at an address which is a multiple of 1024.

2.1 [1 point]: Determine the hit rate for L1 and L2 cache for the execution of update_positions() function. ¿Which is the global hit rate?

2.2 [1 point]: Modify the source code by applying the array merge optimization.

2.3 [0.5 points]: Repeat computations from 2.1 for the resulting code in 2.2.

2.4 [0.5 points]: Assume that a hit at L1 requires 4 cycles and a hit at L2 requires 14 cycles. If the penalty for bringing a block from main memory to L2 cache is 80 cycles, which is the average access time for both cases (2.1 and 2.2)?

2.1: Loop access patter is:

vx[i], x[i], x[i], vy[i], y[i], y[i], vz[i], z[i], z[i], …

Each array has 2^{20} locations of 8 bytes each one. This gives a total of 8 MB per array. Each cache line has 64 bytes, allowing to store 8 array elements.

Cache L1 has 4 ways. Consequently each way is 8KB (2^{13} bytes), giving $2^{13}/2^{6} = 2^{7}$ sets.

As each array has 2^{23} bytes (multiple of way size), each position *I* of all arrays correspondes with the same set in cache. Sequence of hits and misses is:

- VX[0] -> Miss. Select set 0.
- X[0] -> Miss. Select set 1.
- X[0] -> Hit.
- VY[0] -> Miss. Select set 2.
- Y[0] -> Miss. Select set 3.
- Y[0] -> Hit.
- VZ[0] -> Miss. Select set 0. VX send out.
- Z[0] -> Miss. Select set 1. X send out.
- Z[0] -> Hit.

When transitioning to position 1, same sequence of hit and misses is repeated. Consequently:

h(L1) = 3 /9 = 1/3

For lelvel 2 cache, only accesses are performed for cases when a miss in L1 happened. For location 0, 6 misses happen, but for locations 1 to 7 there are 6 hits per location.

h(L2) = 42 / 48 = 7/8

Global hit ratio *h* is:

H = 1 – M

Where:

M = m(h1) * m (h2) = (2/3) * (1/8) = 1/12

H = 11/12

2.2

```
const unsigned int max = 1024 * 1024;
struct part {
  double x, y, z, vx, vy, vz;
};
part vec[max];


void update_positions(double dt) {
  for (unsinged int i=0; i<max; ++i) {
    vec[i].x = vec[i].vx * dt + vec[i].x;
    vec[i].y = vec[i].vy * dt + vec[i].y;
    vec[i].z = vec[i].vz * dt + vec[i].z;
  }
}
```

2.3: Access pattern is:

vec[i].vx, vec[i].x, vec[i].x, vec[i].vy, vec[i].y, vec[i].y, vec[i]. vz, vec[i].z, vec[i]. z, …

Each array location uses 6 locations and a cache entry contains 8 values, in 3 cache lines 4 full array positions fit.

- Location 0 produces 1 misses y 8 hits.
- Location 1 produces 2 hits, 1 miss and 6 hits.
- Location 2 produces 4 hits, 1 miss y 4 hits.
- Location 3 produces 9 hits.

This patter repeats once and again. Consequently:

h(L1) = 33/36 = 11/12

For L2 cache now all misses in L1 cache, are also L2 misses. Thus:

h(L2) = 0

And global ratio:

H = 1 – M

M = m(h1) * m(h2) = 1/12

H = 11/12

2.4: For the original case:

T = 4 + (2/3) * (14 + (1/8) * 80) = 4 + (2/3) * 24 = 4 +16 = 20

For the second case:

T = 4 + (1/12) * (14 + 1 * 80) = 4 + (1/12) *94 = 4 + 7.83 = 11.83