Advanced Computer Architecture CMSC 611

Extra Credit HW1

Due in class at 1.05pm, Nov 14th, 2012

(If you wish to **go green,** then you can submit the entire Homework electronically. Make sure you include the string "CMSC 611 Homework" in your subject line. Deadline remains the same) Please **DO NOT** email your homework to Dr. Olano!! **DO NOT** include him in the CC either!! There is a strong chance it won't be graded if you do!! **Send it only to <abhay1@umbc.edu>**

(The values might not resemble realistic values; it is for the sake of understanding)

1) Parallelization decreases the execution time of a program significantly. Assume we have a program, 36% of which is parallelizable.

   a) What is the theoretical maximum speedup that can be achieved by only exploiting the parallelism?

   $1 / (0.64 + 0.36/\text{infinity}) = 1.56$

   b) Suppose we have a GPU (let's call this GPU-8) which can exploit the parallelism and is 8 times faster than a normal CPU, what is the speedup achieved by using GPU-8?

   $1 / (0.64 + 0.36/8) = 1.46$

   c) Now, instead of GPU-8 used in b) assume a new GPU (let's call this GPU-20), which is 20 times faster than the CPU, but harder to effectively parallelize. What fraction of the original code must be parallelized to match the speedup of GPU-8. (30 points)

   $1.46 = 1 / (1- x + x/20) = 1/ 1 – x +0.05x = 1/ 1 – 0.95x$

   $1 – 0.95x = 0.685$

   $x = 0.3315$ or 33.15%

2) For some program, 35% of the instructions are floating point instructions and 30% are memory instructions. You have 2 possible enhancements, A and B. Enhancement A speeds up the memory instructions by 6 whereas B speeds up the floating point instructions by 10 but also slows down the memory instructions.
   Assume you picked B! Now what is the maximum tolerable slowdown of the memory instructions for you to be able to successfully defend your choice of B as the right one? (15 points)

   Tolerable slowdown is when speed of B is at least as much as A
   $1 / (0.7 + 0.30/6) = 1/ (0.35 + 0.35/10 + 0.3* x)$

Solve for x

(0.7 + 0.30/6) = (0.35 + 0.35/10 + 0.3* x)

0.75 = 0.385 + 0.3x

x = 1.21 is the maximum tolerable slowdown

3) So far our RISC specs allows for register to register ALU instructions only. The operand had to be loaded into the register before the ALU operation. For example,

```
LD      r0, 16(r1)
SUB     r3, r2, r0
```

Now if we supported register-memory addressing mode then we could effectively get rid of the load instruction and combine them into one instruction

```
SUB     r3, r2, 16(r1)
```

For a machine supporting this, the cycle time bumps up by about 15% but CPI remains unaffected. Now assuming that "load" accounts for 35% of the instructions, what minimum fraction of these must be removed, for our new machine to have at-least the same performance as the old one. (30 points)

CPUtimeoriginal = instructions/program * cycles/instruction * time/cycle

CPU time new = (1 - .35 * frac_loads_eliminated) * cycles/instruction * (1+.15) time/cycle

(1 - .35 * frac_loads_eliminated) * cycles/instruction * (1+.15) time/cycle ≤ CPUtimeoriginal

=> (1 - .35* frac_loads_eliminated) * (1.15) ≤ 1

=> 1 - .35* frac_loads_eliminated ≤ 1 / 1.15

=> 1 – 1 / 1.15 ≤ .35* frac_loads_eliminated

=> frac_loads_eliminated ≥ 0.37 or 37.2%

4) (25 points)

Consider the following architectures. Fig. (A) resolves the branch at EX while Fig (B) does it at ID.

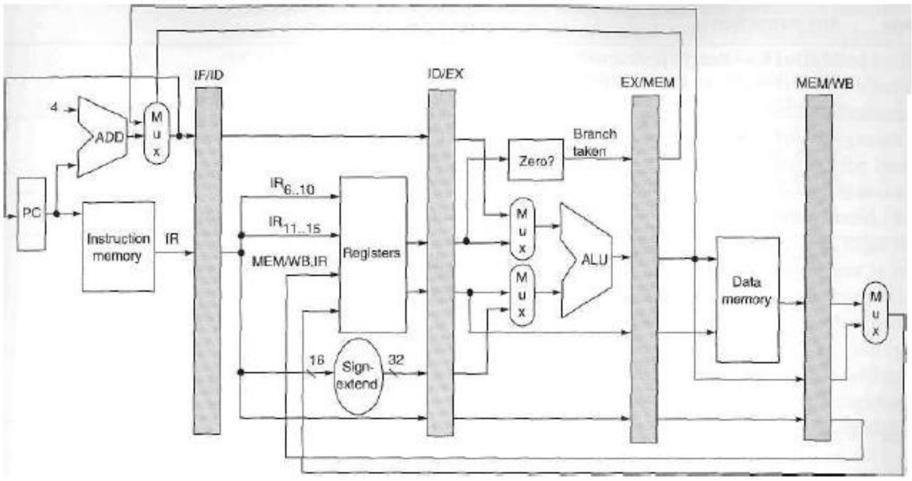

Fig (A)

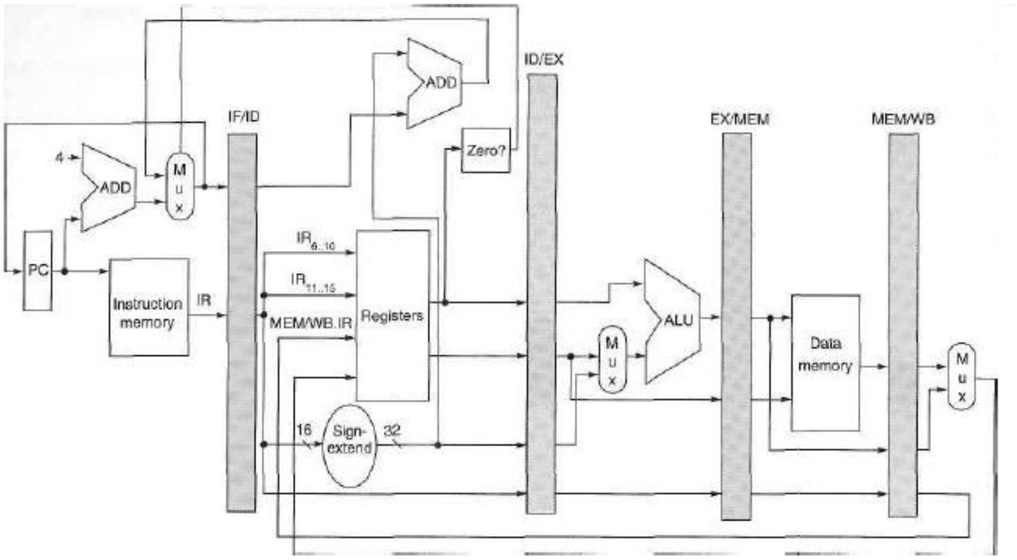

Fig (B)

Now consider a piece of code, 25% of which are branch instructions and 2 machines, A and B, which implement architectures from Fig A and Fig B respectively. Machine A employs a predict-branch-not-taken scheme while Machine B uses one branch delay slot and manages to fill them 70% of the time with useful computation. Assuming that both machine A and B have the same cycle time and that about 35% of the branches are taken, which machine do you think is faster and why?

For the first machine, for 40% of the branches, a cycle is wasted due to the inability to fill the delay slot. The CPI of this machine is 1 + 15% * 40% * 1= 1.06.

Machine A implements a predict not taken. So out of the 25% branches, 35% causes a penalty of 2 cycles (since its at EX stage)
CPI = 1 + 0.25*0.35 * 2 = 1.175

Machine B fills 70% of the 25% branches with useful computation leaving 30% causing a penalty of 1.
CPI = 1 + 0.25 * 0.3 * 1 = 1.075

Therefore, the machine B is faster.