CS 2410

Computer Architecture

Spring 2022

Distributed: Jan 31st, 2022

Due: 11:59pm Feb 9th, 2022

Points: 100

1. (10 pts) Computer A executes the MIPS ISA and computer B executes the x86 ISA. On average, programs execute 1.5 times as many MIPS instructions as x86 instructions. Computer A has an average CPI of 1.5 and computer B an average CPI of 3. If computer B runs at a 3GHz clock frequency, what speed does computer A have to run at to be at least as fast as computer B?

IC_A*CPI_A/CR_A=IC_B*CPI_B/CR_B

1.5*1.5/CR_A=1*3/3GHz

CR_A=2.25GHz

2. (10 pts) Consider two different implementations, M1 and M2, of the same instruction set. There are three classes of instructions (A, B, and C) in the instruction set. M1 has a clock rate of 800MHz and M2 has a clock rate of 1GHz. The average number of cycles for each instruction class and their frequencies (for a typical program) are given in the table below.

Instruction Class	M1 Cycles/Instr Class	M2 Cycles/Instr Class	Frequency of Instr Class
A	1	2	50%
B	2	4	40%
C	4	4	10%

(a) Calculate the average CPI for each machine, M1 and M2.

For M1:

Average CPI=1*50%+2*40%+4*10%=1.7

For M2:

Average CPI=2*50%+4*40%+4*10%=3

(b) Calculate the average MIPS ratings (=clock rate/(CPI*10^6) for each machine, M1 and M2.

For M1:

Average MIPS rating=CR/(CPI*10^6)

=800*10^6/(1.7*10^6)

=470.59

For M2:

Average MIPS rating=CR/(CPI*10^6)

=1000*10^6/(3*10^6)

=333.33

(c) Which machine has a smaller MIPS rating? Which individual instruction class's CPI would you need to change, and by how much, to have this machine have the same or better performance as the machine with the higher MIPS rating (you can only change the CPI for one of the instruction classes on the slower machine)?

M2 has a smaller MIPS rating.

The MIPS rating for M2 can be improved by changing the CPI of instruction class B for Machine M2 to 1.

Modified Average CPI=2*50%+1*40%+4*10%=1.8

For M2: Average MIPS rating=CR/(CPI*10^6)=1000*10^6/(1.8*10^6)

=556 > M1 (471)

3. (20 pts) In practice, applying an optimization to one part of a codebase can require overheads that did not exist in the original execution or even have negative impacts on logically unrelated portions of the code. Consider an already parallel code whose execution consists of SERIAL (S), PARALLEL-COMPUTE (PC), and PARALLEL-SYNC (PS) regions, where, with 1 thread, S=20%, PC=75%, and PS=5% of execution time. Assume that the amount of execution required for PS grows linearly with the number of threads, but is itself 80% parallelizable, and that PC

receives a 1:1 linear speedup with the number of threads. Find 1) the number of threads that maximizes the performance of this code, and 2) the speedup of a 100% parallelizable PS over 1) for the same number of threads that maximizes the performance of 1). (note – no knowledge of parallel threading or synchronization is required to solve this problem – it is a direct application of Amdahl's law)

1)

Assume n represents the # of threads:

Execution time before enhancement = 0.2 + 0.75 + 0.05 = 1

Execution time after enhancement = 0.2 + 0.75/n + 0.05 * 0.2 * n + 0.05 * 0.8 * n/n

Speedup = Execution time before enhancement / Execution time after enhancement

We calculate ds/dn = 0 and get when n = 9, the performance could be maximized.

2)

Speed up = $\frac{0.2+\frac{0.75}{9}+0.05*0.2*9+0.05*0.8}{0.2+\frac{0.75}{9}+0.05} = 1.24$

4. (20 pts) Recall our on-class discussion of different addressing modes. Among all the nine different addressing modes, which addressing mode(s) is most suitable for the following program variables and operations, and why? (Note that you may list multiple addressing modes for a particular variable/operation type, as long as you can justify the reasons)

a. local variables

b. stack variables

c. array elements

d. pointers

e. branch addresses

f. branch condition evaluation.

a. Displacement
Local variables are on the stack and its address could be obtained by using the starting address on the stack with displacement.
b. Auto-decrement and auto-decrement

When push or pop data from the stack, the stack pointer is incremented or decremented.
c. Indexed and scaled

When the start of the array is stored in a register, we can pick the elements by using an index.
d. Register Indirect and memory-indirect

Depend on the pointer stored in register or memory. The value in the register is the address and we use this address in the register to access the value.
e. Relative and displacement

The branch address need add an offset to the current program counter
f. Register

The branch condition evaluation compares two values stored in the register.

5. (20 pts). Using the multi-cycle in order pipeline and its latencies we discussed in slides 51 in 4-reviewPipeline.pdf. Further assume there are two register read ports, one write port, and separate I$ and D$ for instruction and data memory accesses. Schedule the following instructions and draw a cycle-wise table similar to slides 53 as we discussed during the class. Mark the data hazard using arrows on the table (also pay attention to structure hazard if any).

fld F4, 0(R2)

fld F5, 8(R2)

fadd.d F6, F4, F5

fmult, F7, F6, F4

fadd.d F8, F7, F6

fsd F8 16(R2)

Insn	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
fld F4, 0(R2)	F	D	E	M	W																	
fld F5, 8(R2)		F	D	E	M	W																
fadd.d F6, F4, F5			F	D	D	E	E	E	E	W												
fmult, F7, F6, F4				F	F	D	D	D	D	E	E	E	E	E	E	E	W					
fadd.d F8, F7, F6						F	F	F	F	D	D	D	D	D	D	D	E	E	E	E	W	
fsd F8 16(R2)										F	F	F	F	F	F	F	D	E	E	E	M	W

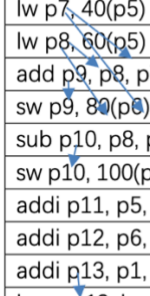
6. (20 pts) Rename the following code segment (don't forget to also show the overwritten register) assuming the initial map table as shown. The free list starts with p7 and has as many free registers in it as you need to complete the renaming (e.g., p7, p8, p9, p10, …). On your

renamed code indicate (by drawing an arrow from the W to the R) how many true data hazards still exist. After renaming, how many instructions have no true data dependencies (and thus could be executed in parallel with enough functional units)? Note that in this example, you are not asked to maintain the free list.

```
loop:  lw $2, 40($6)
       lw $3, 60($6)
       add $4, $3, $2
       sw $4, 80($7)
       sub $4, $3, $2
       sw $4, 100($7)
       addi $6, $6, 4
       addi $7, $7, 4
       addi $1, $1, 1
       bnez $1, loop
```

$1	p1
$2	p2
$3	p3
$4	p4
$6	p5
$7	p6

	Over-written Reg
lw p7, 40(p5)	p2
lw p8, 60(p5)	p3
add p9, p8, p7	p4
sw p9, 80(p6)	
sub p10, p8, p7	p9
sw p10, 100(p6)	
addi p11, p5, 4	p5
addi p12, p6, 4	p6
addi p13, p1, 1	p1
bnez p13, loop	



After renaming, 7 true hazards still exist, 5 instructions have no true data dependencies.