University of Delaware
CISC260 Homework 6 Solution

Jiefu Li

May 12, 2018

1 Problem 1

Answer:

Executable file hearder		
	text size	0x440
	data size	0x90
Text Segement	Address	Instruction
	0x0040,0000	LDR r0, [r3,#0]
	0x0040,0004	ORR r1, r0, #0
	0x0040,0008	BL 0x0000,0130
	...	
	0x0040,0140	STR r0, [r3,#0]
	0x0040,0144	B 0x0000,0174
	...	
	0x0040,0180	MOV pc, lr
	...	
Data Segment	Address	Instruction
	X	0x1000,0000
	...	
	Y	0x1000,0040
	...	

Since procedure A has a text size of 0x140,data size of 0x40 and Procedure B has a text size of 0x300 and data size of 0x50. Therefore, the total text size is 0x440, and total data size is 0x90.

For the given memory allocation strategy, Text starts at 0x0040,0000 and data starts at 0x1000,0000.

All addresses for branching is calculated as following:
destination address - (current address +8).
For example:
for the 3rd instruction, current address is 0x0040,0008, target address is 0x0040,0140 (since the dependency is procedure B), then $0x0040,0140 - (0x0040,0008 + 8) = 0x0040,0140 - 0x0040,0010 = 0x0000,0130$

2 Problem 2

Given processor:
CPI of arithmetic instructions is 1
CPI of load/store instructions is 10

CPI of branch instructions is 3.
Assume a program has 500 million arithmetic instructions 300 million load/store instructions, 100 million branch instructions.

Answer:
a.
Before changing, the program on the old processor needs:
$1 \times 500 + 10 \times 300 + 3 \times 100 = 500 + 3000 + 300 = 3800$ million clock cycles.
Let's assume that for old processor, each cycle's time is 1 (disregarding the unit), then, total time needs is 3800 million.
After improvements:
$1 \times 500.75 + 10 \times 300 + 3 \times 100 = 375 + 3000 + 300 = 3675$ million clock cycles.
Since the clock cycle time is increased by 10%, the total time needs is:
$1.1 \times 3675 = 4042.5$ million
Comparing with 3800 million, 4042.5 million is greater. Therefore, it is not a good choice.

b.
If double the performance of arithmetic, then its CPI is 0.5.
Do same procedure as above to calculate total time needs:
$0.5 \times 500 + 10 \times 300 + 3 \times 100 = 250 + 3000 + 300 = 3550$ million clock cycles.
The overall improvement is: $(3800 - 3550)/3800 = 6.58\%$. If applying Amdahl's law, the procedure is the following:
Proportion of the improvement is: $500/(500 + 3000 + 300) = 500/3800 = 5/38$ $\frac{1}{(1-5/38)+(5/38)/2} = 1.07$, the improvement will be: $1 - 1/1.07 = 6.58\%$
If the performance of arithmetic is improved by 10 times, then its CPI is 0.5.
Do same procedure as above to calculate total time needs:
$0.1 \times 500 + 10 \times 300 + 3 \times 100 = 50 + 3000 + 300 = 3350$ million clock cycles.
The overall improvement is: $(3800 - 3350)/3800 = 11.84\%$. If applying Amdahl's law, the procedure is the following:
Proportion of the improvement is: $500/(500 + 3000 + 300) = 500/3800 = 5/38$ $\frac{1}{(1-5/38)+(5/38)/10} = 38/33.5$, the improvement will be: $1 - 1/(38/33.5) = 11.84\%$
Due to round off, other solutions also considered as correct such as 11.3% for part b.

3 Problem 3

For a given program:
70% of are arithmetic, 10% are load/store, and 20% are branch.

Answer:
a.
Assume that an arithmetic instruction requires two cycles, a load/store instruction takes six cycles, and a branch instruction takes three cycles, find the average CPI.

$2 \times .7 + 6 \times .1 + 3 \times .2 = 2.6$ cycles per instruction.

b
For a 25% improvement in performance, how many cycles, on average, may an arithmetic instruction take if load/store and branch instructions are not improved at all?

If 25% improvements, the the average CPI is $2.6 \times .75 = 1.95$ $x \times .7 + 6 \times .1 + 3 \times .2 = 1.95$ cycles per instruction.
Solve for x, we get $x = 1.07$
c
For a 50% improvement in performance, how many cycles, on average, may an arithmetic instruction

take if load/store and branch instructions are not improved at all?

If 50% improvements, the the average CPI is $2.6 \times .50 = 1.3$ $x \times .7 + 6 \times .1 + 3 \times .2 = 1.3$ cycles per instruction.
Solve for x, we get $x = 0.143$

4 Problem 4

Answer:
See code attached. And here are some explanations.

It can be easily achieved by noticing the following:
If it starts from even, the later operation is + - - + − +...
If it starts from odd, the later operation is - - + − +...
Then, we can define four states:
+ s1(s=1), - s2(s=2), - s3(s=3), + s4(s=4)
And the transition of these four states are:
s1 – s2 – s3 – s4 – s1 – s2...