CALIFORNIA POLYTECHNIC STATE UNIVERSITY
Computer Engineering Program

CPE 315 Computer Architecture Sample Final

Final Exam Solutions

1. **Performance** You are to improve performance on an existing system based on a single-chip microprocessor having the following parameters.

Instruction Type	**% Dyn. Instructions**	**Cycles**
Int	30	1
FP	40	4
Memory	20	5
Branch	10	3

(a) (5 points) What is the average CPI?

Solution: $\text{CPI} = \sum \text{CPI}_i \times \text{ICI}$, where CPI is cycles per instruction, and IC is instruction count.
$\text{CPI} = .3 \times 1 + .4 \times 4 + .2 \times 5 + .1 \times 3 = 3.2$

(b) (8 points) By using an optimizing compiler, the number of FP instructions is reduced by 25% and Memory instructions by 50%. What is the new average CPI when using the optimizing compiler? *Note the instruction mix is changed!*

Solution: The instruction mix has changed, so there is now a different proportion of all instructions. Therefore, we need to recalculate CPI. The total instruction mix has been reduced by 20% (10% due to each of the reductions).
$\text{CPI} = \frac{5(.3\times1+.3\times4+.1\times5+.1\times3)}{4} = 2.875$

(c) (5 points) How much speedup do you expect from using the optimizing compiler in Part (b) compared to the unoptimized version in Part (a)?

Solution: The instruction count is smaller with the new compiler.

$$\begin{aligned} \text{Speedup} &= \frac{\text{time}_{\text{old}}}{\text{time}_{\text{new}}} \\ &= \frac{\text{Instr}_{\text{old}} \times \text{CPI}_{\text{old}} \times \text{CCT}_{\text{old}}}{\text{Instr}_{\text{new}} \times \text{CPI}_{\text{new}} \times \text{CCT}_{\text{new}}} \\ &= \frac{I}{0.8I} \times \frac{3.2}{2.875} \times 1 \\ &= 1.39 \end{aligned}$$

The optimizing compiler improves performance by 39%.

(d) (7 points) If the Memory unit can be made twice as fast, what is the speedup of this program (the clock speed is unchanged) compared to the original (non-optimized, Part (a)) version of the code? *Hint: Use Amdahl's Law, and F ≠20%.*

Solution: Compute F. $F = \frac{.2\times5}{\text{CPI}} = 31.25\%$
$\text{Speedup} = \frac{1}{(1-F)+\frac{F}{S}}$
$\text{Speedup} = \frac{1}{(1-0.3125)+\frac{0.3125}{2}} = 1.185$

2. **ARM ISA and Register Conventions** Write the equivalent ARM Thumb code for the C function below. You must follow correct ARM register usage convention. You should not make any assumptions about the functions func1 and func2 other than they also follow correct register usage convention. Your solution may only use registers `$r0`–`$r7`, `$lr` and `$pc`, and should be less than 25 instructions. Comment your code to receive full credit. Specifically, describe what each register you use represents.

   For the `reorder` function, parameters `a` and `b` are the start addresses of arrays, each with `size` integers. Note that even though this function doesn't return a value, modifications to the values in `a` and `b` will be seen by the calling function.

```
int func1 (int a, int b);
int func2 (int a, int b);

void reorder (int *a, int *b, int size)
{
  for (int i=0; i<size; i++) {
    int num1 = a[i], num2 = b[i];
    a[i] = func1(num1, num2);
    b[i] = func2(num1, num2);
  }
}
```

Solution:

```
        .text
        .global reorder
reorder:
        push    {r4, r5, r6, r7, lr}
        sub     r7, r2, #0
        cmp     r7, #0
        ble     end
        sub     r5, r0, #4
        sub     r4, r1, #4
        mov     r6, #0
loop:
        ldr     r0, [r5, #4]
        ldr     r1, [r4, #4]
        bl      func1
        str     r0, [r5, #4]!
        ldr     r0, [r5, #4]
        ldr     r1, [r4, #4]
        bl      func2
        str     r0, [r4, #4]!
        add     r6, r6, #1
        cmp     r7, r6
        bne     loop
end:
        pop     {r4, r5, r6, r7, pc}
```

3. **Caches**

   (a) For a 512KB 16-way set associative cache that uses a 32-bit address and 32 bytes per line. Calculate the following:

      i. (2 points) How many bits are used for the byte offset?

Solution: 5 bits

ii. (5 points) How many bits are used for the index field?

Solution: 10 bits

iii. (2 points) How many bits are used for the tag?

Solution: 17 bits

(b) (16 points) Consider a 64-byte, direct-mapped cache with an 8-byte block size. For the following access pattern of byte addresses, show the final contents of the tag fields in the cache, and determine if each access is a hit or a miss.

Address:	0x24	0x32	0x18	0x08	0x10	0x11	0x12	0x20
Binary:	00100100	00110010	00011000	00001000	00010000	00010001	00010010	00100000
Hit or Miss?	M	M	M	M	M	H	H	H

Index	**Tag**
000	
001	00
010	00
011	00
100	00
101	
110	00
111	

4. **AMAT** Show all your work.

(a) (10 points) Compute the AMAT of the following cache system:

L1 Has an access time of 3 cycles, and hits 92% of the time.

L2 Has an access time of 15 cycles, and hits 97% of the time.

L3 Has an access time of 30 cycles, and hits 99.8% of the time.

Main Memory has an access time of 200 cycles.

The access time is required for all accesses, hit or miss.

Solution: $AMAT = HT + MR \times MP$
$= 3 + 0.08 \times (15 + 0.03 \times (30 + 0.002 \times 200))$
$= 4.27$

(b) (10 points) If an I-cache has a 95% hit rate, a hit time of 3 cycles, and a D-cache has a 85% hit rate and a hit time of 3 cycles, and both caches have a miss penalty of 40 cycles, what is average memory access time for a program with 25% memory access instructions (e.g., loads, stores, pushes, pops, etc.)?

Solution: $AMAT_I = HT + MR \times MP$
$= 3 + .05 \times 40 = 5$
$AMAT_D = HT + MR \times MP$
$= 3 + .15 \times 40 = 9$

AMAT$_{Total} = \frac{1}{1.25} \times AMAT_I + \frac{.25}{1.25} \times AMAT_D$
$= 5.8$cycles

5. **Pipelines**

(a) For this part, you'll be examining the following ARM code. For all these questions, you may ignore the fill-cost for the pipeline.

```
loop:   add     r2, r2, #1
        ldr     r0, [r3]
        sub     r0, r0, #2
        sub     r1, r3, #4
        ldr     r1, [r1]
        lsl     r1, r1, #1
        add     r1, r0, r1
        str     r1, [r3]
        add     r3, r3, #4
        cmp     r2, r4
        blt     loop
```

i. (6 points) If this loop is executed exactly 10 times, what is the CPI for the loop? Assume the branch is dynamically predicted to be taken in all iterations, and that a mispredicted branch takes 6 cycles.

Solution: loop cycles $= (1_{add} + 4_{ldr+sub} + 6_{sub+ldr+lsl} + 1_{add} + 1_{str} + 1_{add} + 1_{cmp})$+branch cycles

$CPI = \frac{(\text{loop cycles}+1)\times 9+\text{loop cycles}+6}{11\times 10}$

$= \frac{16\times 9+21}{110} = 1.5$

ii. (4 points) Repeat the above calculation if this loop is executed exactly 100 times.

Solution: loop cycles $= (1_{add} + 4_{ldr+sub} + 6_{sub+ldr+lsl} + 1_{add} + 1_{str} + 1_{add} + 1_{cmp})$+branch cycles

$CPI = \frac{(\text{loop cycles}+1)\times 99+\text{loop cycles}+6}{11\times 100}$

$= \frac{16\times 99+21}{1100} = 1.459$

iii. (15 points) Modify the loop only by reordering instructions to minimize CPI. Compute what the minimized CPI is for 10 iterations.

Solution: There are only enough independent instructions to satisfy the latency of one of the load instructions.

```
loop:   sub     r1, r3, #4
        ldr     r0, [r3]
        ldr     r1, [r1]
        add     r2, r2, #1
        sub     r0, r0, #2
        lsl     r1, r1, #1
        add     r1, r0, r1
        str     r1, [r3]
        add     r3, r3, #4
        cmp     r2, r4
        blt     loop
```

loop cycles $= (3_{sub+ldr+ldr} + 4_{ldr+ldr+add+sub} - 2_{ldr+ldr} + 1_{lsl} + 1_{add} + 1_{str} + 1_{add} + 1_{cmp})$+branch cycles

$CPI = \frac{(\text{loop cycles}+1)\times 9+\text{loop cycles}+6}{11\times 10}$

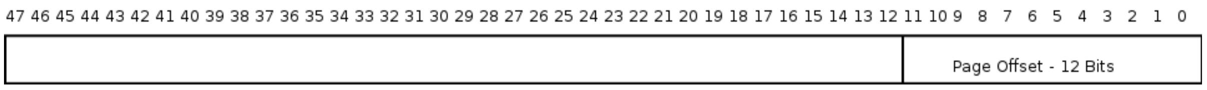
$= \frac{11\times 9+16}{110} = 1.045$

6. **Virtual Memory** In class we learned about how to implement a hierarchical page table structure to allow all page translation and mappings for 32-bit addresses to also be contained in physical pages. However, the lab

workstations are 64-bit systems, and use a 48-bit virtual address (the most significant 16 bits are not used for memory addressing), with 4 KByte pages. With a 64-bit system like the lab machines, *each entry in a page table or page directory will be 64-bits, or 8 bytes.*

(a) (16 points) **Show how to implement a hierarchical page translation system with a 48-bit address** such that all page translations and mappings fit exactly into 4 KByte pages (e.g., all tables, directories, etc. occupy exactly one page).

Show how/where the 48-bit address is divided, and **draw a basic diagram** that illustrates how the hierarchical system works. You do not need to include any TLBs in this diagram.



Solution: 9-bits for each level of the hierarchy: Four tables of lookup before the actual page.

(b) (4 points) In the worst case, how many accesses to physical memory must be made for each memory access?