

NAME: _____ **KEY** _____ UFID[5:8]: _____

Please observe standards of honesty when working on this exam. You should have on your desk your exam booklet, crib sheet, pencils/pens, and calculator.

I certify that I have not received any assistance on this examination except clarifying comments from the instructor, and all work presented here is my own.

Signature: _____

_____ (Students, please do not write below this line) _____

PROBLEM SCORES:

TOTAL SCORE:

Q1___ Q2___ Q3___ Q4___ Q5___ Q6___ Q7___ Q8___ _____ / 200

1 (30 pts) Explain the difference between RISC and CISC in terms of the following:

a) Compare (1-2 sentences) RISC and CISC instruction decoding [15 pts]:

Instruction format:

RISC has variable instruction format, CISC has more regular instruction format.

Design (simplicity, regularity) of hardware:

CISC hardware (HW) difficult to design because of upward compatibility for hundreds of complex instructions, as well as complex instruction decoding mechanism. RISC hardware easier to design, runs faster and tends to consume less power than CISC, due in large part to simplicity of RISC ISA including simple, orthogonal instruction format.

b) Compare register set design in RISC versus CISC [15 pts]:

Number of registers RISC tends to have larger number of registers than CISC

Register length (size) and diversity CISC registers have variable size, RISC registers tend toward fixed size. CISC registers have different lengths, whereas RISC registers are provided mainly in one length (e.g., 32 bits in MIPS)

Other issues: CISC processors tend to run hotter than RISC – more power consumption due to higher CPI in RISC.

2 (45 pts) Multiply the following numbers, and show all your work:

a) Multiply two's complement numbers 11010101_{two} and 01001101_{two} using pencil-and-paper methods (you will have to convert to sign-magnitude, then multiply, then convert back to two's complement): [15 pts]

Pencil&Paper:

00101011	←	11010101 _{two}
<u>x 01001101</u>	←	01001101 _{two}
00101011		
00000000		
00101011		
00101011		
00000000		
00000000		
00101011		
<u>00000000</u>		
00110011101111		

- b) On the next page, multiply the numbers in Part a), above using Booth's algorithm, showing all the steps with all the registers (M,A,Q,Q₋₁) [30 pts]:

Booth's Algorithm: $11010101_{\text{two}} \times 01001101_{\text{two}}$ (numbers in two's complement)

M	A	Q	Q ₋₁	Description
010101	00000000	01001101	0	
010101	00101011	01001101	0	A = A – M
:	00010101	10100110	1	Shift
:	11101010	10100110	1	A = A + M
	11110101	01010011	0	Shift
	00100000	00101001	1	A = A – M
	00010000	00101001	1	Shift
	00001000	00010100	1	Shift
	11011101	00010100	1	A = A + M
	11101110	10001010	0	Shift
	11110111	01000101	0	Shift
	00100010	01000101	0	A = A – M
	00010001	00100010	1	Shift
	11100110	00100010	1	A = A + M
	11110011	00010001	0	Shift

- 3** (25 pts) Answer in salient detail the following questions about pointers and arrays. Do not write code only – answer the questions in plain English, with a high-level language code fragment (e.g., one or two lines) as an example:

- a) What is a pointer, and what does it do? [5 pts]

A pointer is a variable that contains an address that points to a memory element.

- b) In the C language, what is the "&" operator called, and what does it do with respect to pointers? (give HLL code example) [10pts]

Referencing operator – gives address of data item:

```
int x ; int ptr p      # declare x an integer, p a pointer
p = &x                # p gets the memory address of x
```

- c) In the C language, what is the "*" operator called, and what does it do with respect to pointers? (give HLL code example) [10pts]

Dereferencing operator – see next page

- Unary operator `*` gives value that pointer points to
 - if `p = &c => *p == 100` (Dereferencing a pointer)
- Dereferencing \Rightarrow data transfer in assembler
 - `... = ... *p ...;` \Rightarrow load
(get value from location pointed to by p)
 - `*p = ...;` \Rightarrow store
(put value into location pointed to by p)

4 (40 pts) Given the HLL pseudocode:

```
int pointer p,q
array a[10]
int j,k
i = 2 ; j = 1; q = a[3]
p = z; q = p + 3
*p = i ;
a = &z[y] ;
*q = j
```

Write (a) memory model diagram that describes the effects of the preceding code (e.g., show j,k, the array a, and the pointers p and q) [10 pts], (b) assumptions [5 pts], and (c) MIPS code [25 pts] that implements the preceding pseudocode.

a) MIPS Memory Model Diagram:

b) Assumptions for your MIPS code

c) Actual MIPS code with comments (no need to do procedure calls):

- 5 (20 pts) Suppose you want to develop a function or subroutine that passes an array of N elements (1 word per element) as an argument. Please answer the following questions, and *tell why you gave your answer*. With no justification, you will not get full credit...

- a) How does *pass by reference* work (give HLL code example)? [6pts]

Pass by reference passes a pointer to a data value to the called or calling procedure. Example:

```
int ptr x,y
int a[100],b[100]
```

```
:
```

```
x = a ; y = b
```

```
result = fun(x,y)
```

```
# x and y get base addrs of a,b
```

```
# instead of calling fun(a,b)
```

- b) Give code example for *pass by value* (HLL code) and tell how it works. [6pts]

Pass by value sends actual value(s) of variable (or array) to called or calling procedure. In previous example, substitute **result = fun(a,b)** for result = fun(x,y)

- c) Which technique (pass by reference or pass by value) is better for functions with scalar (non-array) arguments? Explain why, to get full credit. [8pts]

Either pass by reference or pass by value works for scalar arguments, because a pointer takes up as much space as a long integer. However, for arrays, pass by reference is more efficient because less data must be put on the stack.

- 6 (20 pts): Compare and contrast the multiplier and divider algorithms *for signed integers* and their hardware implementation, re: amount of hardware, shifting operations, and estimated execution time for processing two N -bit numbers.

Reference the answer to Problem 6 on Homework #3.

- 7 (20 pts) **Amdahl's Law:** Suppose a computer has a floating-point unit. How fast does the floating-point unit need to be accelerated (e.g., 2 times faster, 3.78 times faster,...) to make the computer's execution speed of a program P 1.2 times faster, where P uses the floating-point unit 50 percent of the time? Show all your work to get full credit...

$$\text{Speedup} = \frac{1}{(1 - \text{fraction enhanced}) + (\text{fraction enhanced}/\text{factor of improvement})}$$

So $1.2 = 1 / [(1 - 0.5) + (0.5 / x)]$ Solve for X to get FP speedup = 1.5X.

- 8 (30 pts **EXTRA CREDIT**) Explain what a multicycle datapath (MCDP) is and what it does, then explain how it differs from a single-cycle datapath (SCDP). Then, tell how MCDP achieves faster execution time than SCDP for the same program.

→ What is a multicycle datapath and what does it do? [10 pts]

A multicycle datapath efficiently performs the work of a computer by dividing the computational cascade into multiple stages, each of which requires one clock cycle to perform.

→ How does MCDP differ from SCDP? [10 pts]

Because the hardware dedicated to each stage is buffered and is much smaller than the hardware for the entire cascade, the settling time per stage is much less than for the single-cycle datapath.

→ How does MCDP achieve faster execution time than SCDP? Discuss hardware issues in detail. [10 pts]

Although this process of subdividing the computational cascade increases CPI, clock cycle time is significantly decreased, yielding more efficient execution. Also, MCDP requires more hardware than SCDP due to buffering between stages.