

**The University of Alabama in Huntsville**  
**Electrical and Computer Engineering Department**  
**CPE 221 01**  
**Test 2**  
**April 12, 2018**

**This test is closed book, closed notes. You may not use a calculator. You should have the 6 page ARM Instruction Reference. Please check to make sure you have all 6 pages of the test. You must show your work to receive full credit.**

Name: \_\_\_\_\_

1. (1 point) A \_\_\_\_\_ points to a place on the stack and can change throughout the lifetime of an instance of a procedure.
2. (1 point) A \_\_\_\_\_ control unit runs a program whose input is the machine-level op-code to be executed and whose output is the bus enables, multiplexer controls, clocks, and signal that control the processor.
3. (1 point) Pipelining is a technique in for improving the \_\_\_\_\_ of instruction execution.
4. (1 point) A 1 address instruction has a register, called the \_\_\_\_\_ to hold one operand and the result.
5. (1 point) \_\_\_\_\_ is a load/store RISC ISA that has 32 general purpose registers.
6. (10 points) A processor executes an instruction in the following four stages. The time required by each stage in picoseconds (1,000 ps = 1 ns) is given for each stage.

IF	Instruction fetch	320 ps
OF	Operand fetch	280 ps
OE	Execute	350 ps
OS	Operand store (writeback)	220 ps

  - a. (5 points) What is the time taken to fully execute an instruction assuming that this structure is pipelined in four stages and that there is an additional 10 ps per stage due to the pipeline latches?
  - b. (5 points) Suppose that 20% of instructions are branch instructions that are taken and cause a 2-cycle penalty, what is the effective instruction execute time?

7. (30 points) Consider the following ARM program. Trace the values of the registers shown as they change during program execution. Also, trace the writes to memory by the STR instruction (non-stack) and the stack activity. Clearly indicate the value of the sp. There may be unused columns or rows in the tables. If you need to add columns or rows, you may do so.

```
; This program multiplies two numbers by repeated addition.
; First, take absolute values of both numbers, do multiplication,
; then adjust result as necessary.

        AREA MULTIPLY_BY_ADDING, CODE, READONLY
        ENTRY
0         LDR    r0, num1          ; Put num1 in r0.
4         LDR    r1, num2          ; Put num2 in r1.
8         MOV    fp, #0x0000C000
12        MOV    sp, #0x00000000
16        ADR    r10, case1
20 mpy_ne MOV    r3, #0             ; Set r3 to 0, it will hold the result.
24        TEQ    r0, #0            ; Compare first parameter to 0
28        BEQ    done             ; If first parameter is 0, done, result = 0.
32        TEQ    r1, #0            ; Compare second parameter to 0
36        BEQ    done             ; If second parameter is 0, done, result = 0.
40        PUSH   {fp}             ; Save old frame pointer before branching
44        MOV    fp, sp           ; Copy stack pointer into the frame pointer
48        SUB    sp, sp, #8        ; Make space on the stack - one input, one output
52        STR    r1, [fp, #-4]     ; Store the input parameter on the stack
56        BL     abs              ; Branch to abs routine and store return in lr
60        LDR    r4, [fp, #-8]     ; Copy result from the stack into r4
64        SUB    sp, sp, #8        ; Make space on the stack - one input, one output
68        STR    r0, [fp, #-4]     ; Store the input parameter on the stack
72        BL     abs              ; Branch to abs routine and store return in lr
76        LDR    r5, [fp, #-8]     ; Copy result from the stack into r5
80        MOV    sp, fp           ; Collapse the frame by moving the stack pointer
84        POP    {fp}             ; Pull the old frame pointer off the stack
88 adding ADD    r3, r3, r5        ; Add num2.
92        SUBS   r4, r4, #1        ; Decrement r4, the abs of num1.
96        BEQ    adjust           ; If r4 = 0, done adding, go to adjust.
100       B      adding           ; Otherwise, need to add again.
104 adjust MOVS   r0, r0           ; Done adding, now adjust sign of result.
108        RSBMI r3, r3, #0        ; If num2 negative, negate result.
112        MOVS   r1, r1
116        RSBMI r3, r3, #0        ; If num1 negative, negate result.
120 done   STR    r3, [r10]        ; Store the result
124 final  B      final
128 abs    LDR    r9, [fp, #-4]     ; Copy the input parameter off of the stack
132        CMP    r9, #0           ; Test input parameter
136        BPL    d_abs           ; If zero or greater, we're done
140        RSB    r9, r9, #0       ; If negative, make it positive
144 d_abs  STR    r9, [fp, #-8]     ; Store the result on the stack
148        MOV    pc, lr           ; Put lr value in pc to return
152 num1   DCD    -3              ; Give num1 a value
156 num2   DCD    2               ; Give num2 a value
160 case1  SPACE  4              ; Make space for result
        END
```

Results of the *STR* instructions not using stack.

Memory Address	Contents

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	

Instruction:

---

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	

Instruction:

---

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	

Instruction:

---

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	

Instruction:

---

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	

Instruction:

---

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	

Instruction:

---

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	

Instruction:

---

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	

Instruction:

---

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	

Instruction:

---

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	

Instruction:

---

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	

Instruction:

---

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	

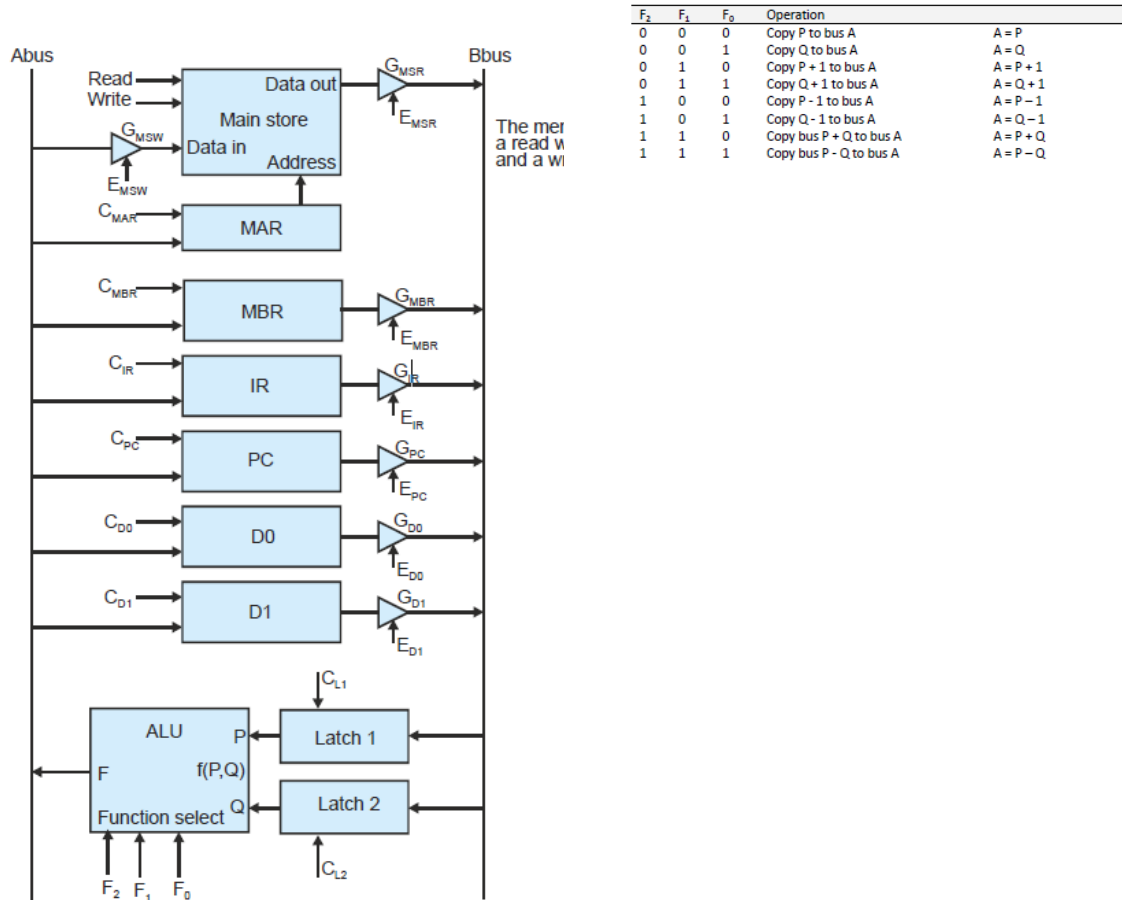
Instruction:

---

r0						
r1						
r3						
r4						
r5						
r9						
r10						
lr						
fp						

8. (20 points) Write the code to implement the expression  $A = (((B/(F - G)) + C) \times D) + E$  on 3-, 2-, 1-, and 0-address machines. Do not rearrange the expression. In accordance with programming language practice, computing the expression should not change the values of its operands. When using a 0-address machine, the order used is SOS op TOS, where SOS is second on stack and TOS is top of stack.

9. (20 points) For the architecture shown, write the sequence of signals and control actions necessary to execute the instruction `STRI (P), D0, D1`, that stores the sum of D0 and D1 in the memory location pointed to by the contents of the memory location P. Assume that the address P is in the instruction register IR. The actions of this instruction are described by the abstract RTL  $M[M[P]] \leftarrow D0 + D1$ .



Cycle	Concrete RTL	Signals
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

10. (15 points) A RISC processor executes the following code. There are data dependencies but no internal forwarding. A source operand cannot be used until it has been written. Assume that the first instruction begins executing in the first cycle.

```
MUL  r0, r1, r2
ADD  r3, r1, r4
ADD  r5, r1, r6
ADD  r6, r0, r7
LDR  r1, [r2]
```

- a. (8 points) Assuming a five-stage pipeline (fetch (IF), operand fetch (OF), execute (E), memory access (M), and register write (W)), what registers are being read during the fifth clock cycle and what register is being written? Are the values read the correct ones? Why or why not?

	1	2	3	4	5	6	7	8	9	10	12	13	14	15	16	17
MUL r0, r1, r2																
ADD r3, r1, r4																
ADD r5, r1, r6																
ADD r6, r0, r7																
LDR r1, [r2]																

- b. (7 points) Assuming an eight-stage pipeline: fetch (F), decode (D), register read (O), execute 1 (E1), execute 2 (E2), memory read (MR), memory write (MW), register write (WB), how long will it take to execute the entire sequence?

	1	2	3	4	5	6	7	8	9	10	12	13	14	15	16	17
MUL r0, r1, r2																
ADD r3, r1, r4																
ADD r5, r1, r6																
ADD r6, r0, r7																
LDR r1, [r2]																