The University of Alabama in Huntsville
Electrical and Computer Engineering Department
CPE 221 01
Test 2 Solution
Fall 2018

This test is closed book, closed notes. You may 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.

1. (1 point) High-level language programmers use _**variables**_ to represent any type of data element they define.
2. (1 point) An _**exception**_ is an event that forces the computer to stop normal processing and to switch control to the operating system.
3. (1 point) _**Pipelining**_ is a technique for improving the throughput of instruction execution.
4. (1 point) A _**frame**_ is a region of temporary storage at the top of the current stack.
5. (1 point) The computer has a _**program counter**_ that contains the address of the next instruction to be executed.
6. (10 points) A processor executes an instruction in the following five stages. The time required by each stage in picoseconds (1,000 ps = 1 ns) is:

IF	Instruction fetch	320 ps
OF	Operand fetch	280 ps
OE	Execute	350 ps
M	Memory	450 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 five stages and that there is an additional 20 ps per stage due to the pipeline latches? **5 * (max(IF, OF, OE, M, OS) + latch) = 5 * (max(320, 280, 350, 450, 220) + 20) ps = 5*470 = 2350 ps**

   b. (5 points) What is the time to execute an instruction if the processor is not pipelined? **1 * (IF + OF + OE + M + OS) = (320 + 280 + 350 + 450 + 220) ps = 1620 ps**

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

```
#include <stdio.h>
int Factorial (int);

int main()
{
  int n = 1;
  int f;
  f = Factorial(n);
}

int Factorial (int n)
{
  if (n == 1)
    return 1;
  else
    return n*Factorial(n-1);
}
```

```
       AREA  FACTORIAL, CODE, READONLY
       ENTRY
main
0         mov    lr, #60
4         mov    fp, #0x0000C000
8         push   {fp}
12        sub    sp, sp, #8
16        add    fp, sp, #0
20        movs   r0, #1
24        bl     factorial
28        str    r0, [fp, #4]
32        movs   r3, #0
36        mov    r0, r3
40        adds   fp, fp, #8
44        mov    sp, fp
48        pop    {fp}
52  stop  b      stop
factorial
56        str    r0, [fp, #-4]
60        ldr    r3, [fp, #-4]
64        cmp    r3, #1
68        bne    L4
72        movs   r3, #1
76        b      L5
L4
80        ldr    r3, [fp, #-4]
84        subs   r3, r3, #1
88        mov    r0, r3
92        bl     factorial
96        mov    r2, r0
100       ldr    r3, [fp, #-4]
104       mul    r4, r3, r2
108       mov    r3, r4
L5
112       mov    r0, r3
116       mov    pc, lr
       END
            END
```

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	0x0000 C000

Inst: _8 push {fp}_

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	0x0000 C000

Inst: _12 sub sp, sp, #8_

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	0x0000 C000

Inst: _16 add fp, sp, #0_

Address	Value
FFFF FFEC	
FFFF FFF0	1
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	0x0000 C000

Inst: _56 str [fp, #-4]_

Address	Value
FFFF FFEC	
FFFF FFF0	1
FFFF FFF4	
FFFF FFF8	1
FFFF FFFC	0x0000 C000

Inst: _28 str r0, [fp, #4]_

Address	Value
FFFF FFEC	
FFFF FFF0	1
FFFF FFF4	
FFFF FFF8	1
FFFF FFFC	0x0000 C000

Inst: _40 adds fp, fp #4_

Address	Value
FFFF FFEC	
FFFF FFF0	1
FFFF FFF4	
FFFF FFF8	1
FFFF FFFC	0x0000 C000

Inst: _44 mov sp, fp_

Address	Value
FFFF FFEC	
FFFF FFF0	1
FFFF FFF4	
FFFF FFF8	1
FFFF FFFC	0x0000 C000

Inst: _48 pop {fp},
sp = 0x0000 0000,
fp = 0x0000 C000_

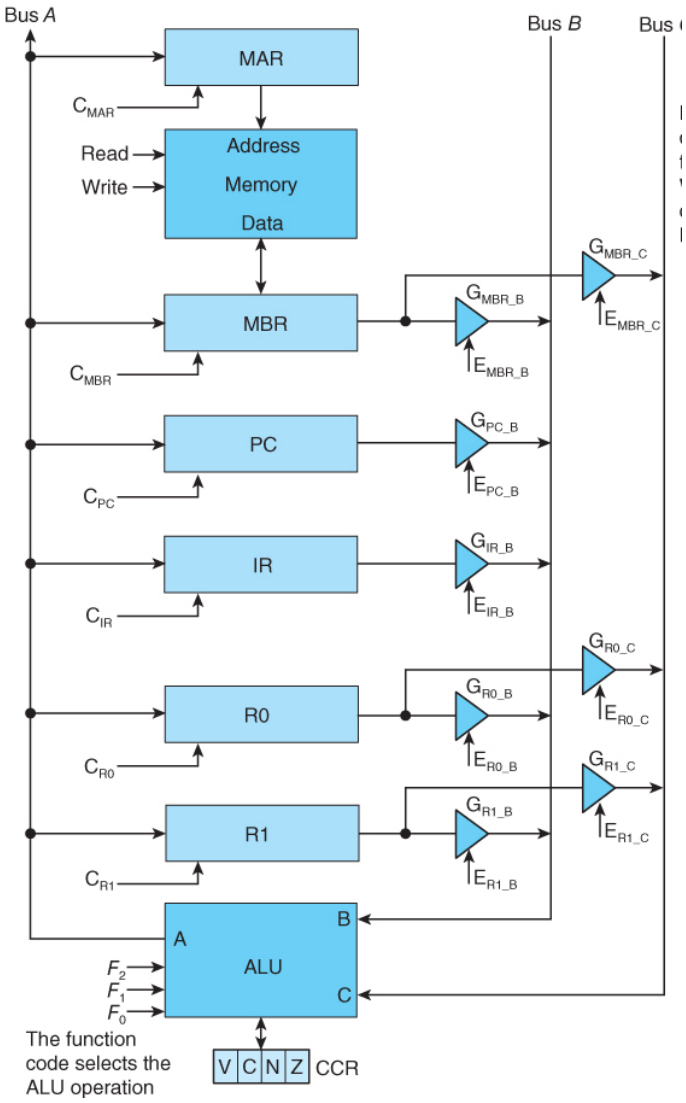
sp indicated in yellow, fp in blue, green indicates sp and fp have the same value

r0	1	1	0			
r1	1	1	0			
r3						
r4						
r5						
r9						
r10						
lr	60	28				
fp	0x0000 C000	0xFFFF FFF4	0xFFFF FFFC	0x0000 C000		

8. (20 points) Write the code to implement the expression A = (B*(F – G))/(C × (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.

3-address	2-address	1-address	0-address
SUB A, F, G	LOAD A, F	LDA D	PUSH B
MUL A, B, A	SUB A, G	ADD E	PUSH F
ADD T, D, E	MPY A, B	MPY C	PUSH G
MPY T, T, C	LOAD T, D	STA T	SUB
DIV A, A, T	ADD T, E	LDA F	MPY
	MPY T, C	SUB G	PUSH C
	DIV A, T	MPY B	PUSH D
		DIV T	PUSH E
		STA A	ADD
			MPY
			DIV
			POP A

9. (20 points) For the architecture shown, write the sequence of signals and control actions necessary to execute the instruction `STRI (P), R0, R1`, that stores the sum of R0 and R1 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]] ← R0 + R1`.



F_2	F_1	F_0	Operation
0	0	0	A = B
0	0	1	A = C
0	1	0	A = B + 1
0	1	1	A = C + 1
1	0	0	A = B – 1
1	0	1	A = C – 1
1	1	0	A = B + C
1	1	1	A = C - B

Cycle	Concrete RTL	Signals
1	MAR ← IR	E_{IR_B}, F = 000, C_{MAR}
2	MBR ← M[MAR]	READ, C_{MBR}
3	MAR ← MBR	E_{MBR_B}, F= 000, C_{MAR}
4	MBR ← R0 + R1	E_{R0_B}, E_{R1_C}, F = 110, C_{MBR}
5	M[MAR] ← MBR	WRITE

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.

```
a  MUL  r0, r1, r2
b  ADD  r3, r1, r4
c  LDR  r1, [r2]
d  ADD  r5, r1, r6
e  ADD  r6, r0, r7
f  STR  r3, [r2]
```

The dependences that exist are:
a-e, b-f, c-d

a. (6 points) Assuming a four-stage pipeline (fetch (IF), operand fetch (OF), execute (E), operand write (W)), what registers are being read during the seventh clock cycle and what register is being written?
`ADD r5, r1, r6` is in the OF stage, reading r1 and r6
No instruction is in the W stage, not writing any register

	1	2	3	4	5	6	7	8	9	10	11	12	13
`MUL r0, r1, r2`	IF	OF	E	W									
`ADD r3, r1, r4`		IF	OF	E	W								
`LDR r1, [r2]`			IF	OF	E	W							
`ADD r5, r1, r6`				IF			OF	E	W				
`ADD r6, r0, r7`							IF	OF	E	W			
`STR r3, [r2]`								IF	OF	E	W		

b. (9 points) Assuming a six-stage pipeline: fetch (F), register read (O), execute (E), memory read (MR), memory write (MW), register write (WB), how long will it take to execute the entire sequence? **15 cycles as indicated in the table.**

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
`MUL r0, r1, r2`	F	O	E	MR	MW	WB									
`ADD r3, r1, r4`		F	O	E	MR	MW	WB								
`LDR r1, [r2]`			F	O	E	MR	MW	WB							
`ADD r5, r1, r6`				F					O	E	MR	MW	WB		
`ADD r6, r0, r7`									F	O	E	MR	MW	WB	
`STR r3, [r2]`										F	O	E	MR	MW	WB