

Name: _____

QUIZ #1

*Closed book except for distributed materials and one 3x5 card with handwritten, original notes. No calculators. All questions are assumed to refer to the ARM7TDMI microprocessor unless otherwise specified. All work is to be your own - **show your work** for maximum partial credit.*

Select only one answer unless specified otherwise.

1) (4pts) Which instruction would cause an assembler error?

- a) ADD R0, R0, #0xFFFFFFFF
- b) ORR R0, R0, #0x80000001
- c) AND R0, R0, #0x8000007F**
- d) none of the above

Can't rotate an odd amount

2) (4pts) Which processor architecture is most suited for the use of self modifying code (code that can change itself)?

- a) a Harvard architecture
- b) a Von Neumann architecture**
- c) a RISC architecture
- d) a CISC architecture

Full instruction set available to work on (modify) program memory

3) (4pts) In an assembly language file, the statement "myVar DCW 0x00A5" will allocate

- a) two bytes**
- b) four bytes
- c) the number of bytes depends on whether it is ARM or Thumb code
- d) no bytes

4) (4pts) The assembler will catch which type(s) of errors? **(circle all correct answers)**

- a) syntax errors**
- b) logic (run-time) errors
- c) missing EXPORT directives
- d) missing IMPORT directives**

Missing IMPORT will be caught by assembler because you will reference something undefined at assembly time. Missing export will be caught by linker.

5) (4pts) For the instruction: ADR R0, myLabel

- a) The address of myLabel is stored as a 24-bit immediate in the instruction encoding.
- b) The assembler will replace this with an instruction of the form:
ADD R0, PC, #xxxx OR SUB R0, PC, #yyyy**
- c) There will be a syntax error generated since there is no such instruction.
- d) None of the above

6) (10pts) Consider the following code fragment for the ADuC7026 used in this class:

```

MOV R0, #0x00010000
MOV R1, #0x01
STRB R1, [R0], #1
LSL R1, #1
STRB R1, [R0], #1
LSL R1, #1
STRB R1, [R0], #1
LSL R1, #1
STRB R1, [R0], #1
LDR R3, [R0, #-1]
    
```

These are post incrementing stores. R0 is updated after the memory access.

a) What data is stored in memory location 0x00010001

0x01

b) What is the final value of R0

0x00010004

c) What is the value in R3 when finished?

0x08

d) What did the assembler do when it encountered the "LSL R1, #1" instruction?

It replaced it with:
MOV R1, R1, LSL #1

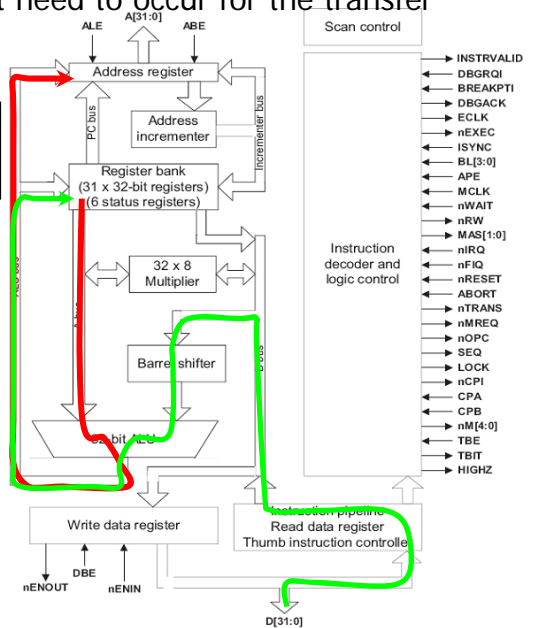
7) (10pts) Explain the datapath for the load instruction. Indicate the path on the block diagram, and explain in words the steps that need to take place.

```
LDR R0, [R1]
```

- R1 gets read and transferred through the ALU to the memory Address register.
- Memory is read: Mem[R0]
- This data comes in through the data bus: D[31:0] and is transferred through the shifter operand path and into R0

R1 to memory address register

Mem[R1] gets transferred to R0



8) (8pts) Debuggers:

a) List 2 advantages a simulation based debug tool has over a JTAG based debug tool.

- **Available before hardware**
- Can input stimulus through files
- Can in some instances be faster or closer to real time trace since don't have the long serial shifting of data in/out.
- Cheap/free

b) List 2 advantages a JTAG based debug tool has over an ICE based debug tool

- **Much Cheaper**
- More available since it isn't a royal pain for the manufacturer to support.
- No special packaging needed for the chip (only uses 4 extra pins), so no different PCB with different footprint needed for debug

9) (6pts) Why is memory allocated in the code area typically initialized, whereas memory allocated in the data area is almost never initialized?

- In the code area it is in Flash (non-volatile memory) so you can initialize constants and have them available.
- In the data area you are dealing with SRAM (volatile). If the programmer was to initialize it wouldn't do much good. The SRAM will lose state when power is removed.

10) (6pts) Why should the main routine of an embedded application always be some form of an infinite loop? (aka: why always end your code with a branch?)

- A processor never stops fetching & executing instructions (unless it happens to have a halt instruction). If your code does not end with a loop, or contain a continuous wait for interrupts loop. Then it will fetch code right off the end of your programmed code space and execute random (potentially harmful) instructions.

11) (10pts) True or False (Circle one):

- a) True / **False** → At power up the ARM will have interrupts enabled.
- b) **True** / False → At power up the ARM will be in supervisory mode.
- c) True / **False** → System mode has the most number of banked (duplicated) registers.
- d) **True** / False → There are no special addressing modes to access MMR's.
- e) True / **False** → The ARM7 defaults to Thumb mode when first powered on.

- 12) (6pts) Using the minimum number of instruction(s) (with **only** register and **legal** immediate operands); write a code fragment that will place the value 0xFFFF00FF in R0. (You cannot use any load instructions or pseudo-instructions.)

```
MVN R0, #0xFFFF00FF
```

Sorry, this wasn't a trick question. 0x000FF000 is an achievable immediate with 8-bits and a rotate. So just need to use the MVN to get its complement.

- 13) (10pts) Given the word-wide memory map (initial, before code is run) and code shown below, answer the following questions:

```
LDR      R0, =(0x12345670)
LDR      R1, [R0, #4]!
STR      R1, [R0, #4]
LDMDB   R0!, {R2, R3}
```

0x12345678	0x00000000
0x12345674	0x11111111
0x12345670	0x22222222
0x1234566C	0x33333333
0x12345668	0x44444444
0x12345664	0x55555555

R0 = 0x____**0x1234566C**_____ (after code executes)

R1 = 0x____**0x11111111**_____ (after code executes)

R2 = 0x____**0x33333333**_____ (after code executes)

R3 = 0x____**0x22222222**_____ (after code executes)

What memory location did the STR instruction write, and to what value?

It wrote to location 0x12345678, and it wrote data: 0x11111111

(14pts) **Memory Map:** A microprocessor system has the byte-wide memory map shown below. *You must show calculations (or explain thought process) to receive credit for your answers.*

How many lines must the address bus have?

The full address of 0x3FFFF would require 18 set bits to access it.

What is the total size of the memory space (in bytes, using appropriate suffix)?

$2^{18} = 2^{16} \times 2^2$. Every geek worth his/her weight in dirt knows $2^{16} = 64\text{kBytes}$
Therefore this must be 256kBytes

What is the size of the ROM (in bytes, using appropriate suffix)?

$0x3FFFF - 0x28000 = 0x17FFF$. Again I know 0xFFFF is 64kBytes. I should also realize 0x7FFF is half of that. So we have 64kBytes + 32kBytes = 96kBytes

What is the address of the last RAM location?

We know it is 32kBytes. We know 64kBytes is an address of 0xFFFF. Therefore it would be half of that address or 0x7FFF.

You were told to have a few basic powers of 2 memorized or written on your 3x5 card.

