CSE 30 Fall 2017 – Midterm 1

Name:

PID:

Email:

Only answers on this sheet will be graded. Per-question points are in italics. There are 70 total points

1. *(3)*	10. *(5)*
2. *(3)*	11. *(5)*
3. *(2)*	12. *(5)*
4. *(2)*	13. *(5)*
5. *(2)*	14. *(5)*
6. *(2)*	15. *(5)*
7. *(2)*	16. *(5)*
8. *(2)*	17. *(5)*
9. *(5)*	18. *(7)*

Instructions If the answer is "None of the above," make sure to answer the appropriate letter for that response. If the answer is a sequence of letters, make sure to give an answer for each, and put them one after another in the answer cell. If an answer says "Write your answer directly in the answer sheet," follow the instructions in the question for the answer. You might write a single constant in a particular format, a short line of code, or several names of instructions or labels.

If you need to write an answer in hexadecimal, prefix it with `0x`.

For questions that are multiple-select (e.g. choose ALL that apply), you gain points both for putting correct answers and lose points for putting incorrect answers. You can't get negative points on a question.

Unless otherwise stated in the question, assume 32-bit words and ARM assembly semantics.

Remember to put your answers in the answer key!

We will not answer questions about the exam during the session. If you believe there is a mistake on the exam that makes a question be nonsense or un-answerable, note it on the question and on your answer sheet by writing "BAD QUESTION," and try to answer the question as best you can.

Number Systems

1. How many representations of 0 are possible in the signed 2's complement binary representation of numbers? Write your answer as a single digit directly in the answer sheet.

2. How many representations of 0 are possible in the signed-magnitude representation of numbers? Write your answer as a single digit directly in the answer sheet.

For each of the following 8-bit, signed, two's complement binary numbers, give the corresponding **decimal** representation, including the negative sign if necessary. Write your answers directly in the answer sheet.

3. `1111 1111`

4. `1111 1001`

5. `0010 0001`

For each of the following decimal numbers, convert them to their unsigned hexadecimal representation, using as many digits as necessary. Write your answers directly in the answer sheet.

6. `168`

7. `512`

8. `17`

Machine Code

Consider these instructions:

A: `subne r10, r10, #4`

B: `subeq r2, r5, r3`

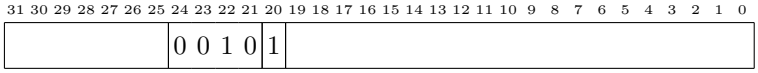
C: `subs r10, r2, r10`

D: `adds r2, r10, r3`

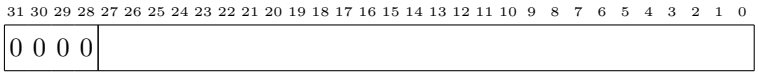
E: `moveq r1, r2`

For each of the next three questions, you are given a a machine instruction with specific bits set, with unknown bits set in the other positions. For each question, choose the instruction(s) above whose machine instruction encoding must have the specified bits set. Each instruction could fit in zero or more categories. Give your answer a a sequence of letters. Choose ALL that apply in each case, and write NONE if none apply.

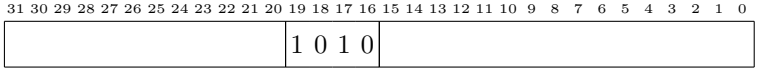
9.



10.



11.



Bitwise Operations, Instructions

Consider this program:

```
mov r2, #0
_____ r1, r2, r1
```

For the next three questions, some property is stated. Consider each of the command choices below filled into the blank in the program. For each answer, give the letter(s) of the commands which would make the program satisfy the property after the two instructions run. The value in r1 is intentionally unknown. Choose ALL that apply.

A: `and`

B: `eor`

C: `orr`

D: `tst`

E: `sub`

12. The condition flags (e.g. the first four bits of `cpsr`) do not change

13. Switching the arguments to instead be `r1, r1, r2` would have the same effects as the original order

14. The value in r1 is always the same as before the two instructions run

15. Which of the following programs has the effect of flipping all of the bits in r1 (by "flip" we mean that all 1's become 0's and vice versa)? Choose ALL that apply.

A:
```
mov r0, #0x0
and r1, r1, r0
```

B:
```
mov r0, #0xFFFFFFFF
eor r1, r1, r0
```

C:
```
mov r0, #0x0
eor r1, r1, r0
```

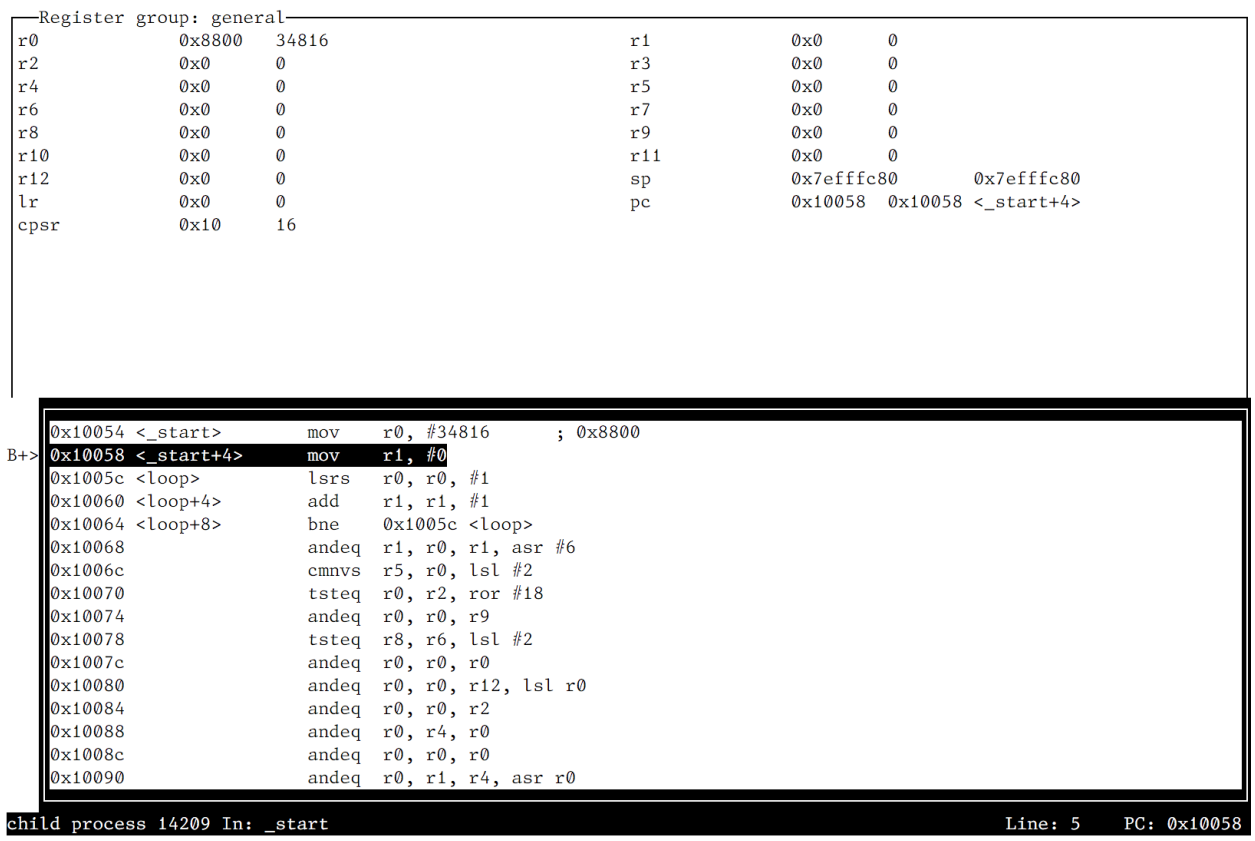
D:
```
mov r0, #0xFFFFFFFF
and r1, r1, r0
```

E:
```
mov r0, #0xFFFFFFFF
orr r1, r1, r0
```

F: None of the above

Debugging, Machine State

Consider this snapshot of a running process in `gdb`:



16. What are the next 4 values that the `pc` will hold **not including** the current one? Choose from the following list; give your answer as a sequence of 4 letters. You may use letters more than once, or not at all.

    A: `0x10054`
    B: `0x10058`
    C: `0x1005c`
    D: `0x10060`
    E: `0x10064`
    F: `0x10068`

17. What value will be in r0 the first time the program counter holds the address `0x10068`, but before that instruction is executed? The given values are in decimal. Give your answer as a single letter.

    A: `1`
    B: `32`
    C: `4`
    D: `16`
    E: Some other value
    F: The program counter will never hold the address `0x10068` due to an infinite loop

Assembly Programming

18. Consider this program with holes in it:

```
mov r2, #0
loop:
  ------------
  ------------
  add r2, #1
skip_add:
  ------------
```

Assume the goal is to have r2 contain the number of (binary) 1's in the value in r1. For example, if r1 contains

`0000 1010 0001 1011 0100 0111 0000 0000`

the value in r2 at the end should be (decimal) `10` or (hex) `0xa`.

The program should always terminate (it should not produce an infinite loop).

Give your answer as a sequence of 3 letters corresponding to instructions below that create a program to this specification when filled in above in order.

A: `lsl r1, #1`
B: `lsls r1, #1`
C: `lsr r1, #1`
D: `lsrs r1, #1`
E: `b skip_add`
F: `bcc skip_add`
G: `bcs skip_add`
H: `beq skip_add`
I: `bne skip_add`
J: `b loop`
K: `bcc loop`
L: `bcs loop`
M: `beq loop`
N: `bne loop`

Figures from Harris et al, *Digital Design and Computer Architecture*, Second Edition. 2016. Appendix B and Chapter 6

Table 6.1 ARM register set

Name	Use
R0	Argument / return value / temporary variable
R1–R3	Argument / temporary variables
R4–R11	Saved variables
R12	Temporary variable
R13 (SP)	Stack Pointer
R14 (LR)	Link Register
R15 (PC)	Program Counter

Table B.5 Instructions that affect condition flags

Type	Instructions	Condition Flags
Add	`ADDS, ADCS`	*N*, *Z*, *C*, *V*
Subtract	`SUBS, SBCS, RSBS, RSCS`	*N*, *Z*, *C*, *V*
Compare	`CMP, CMN`	*N*, *Z*, *C*, *V*
Shifts	`ASRS, LSLS, LSRS, RORS, RRXS`	*N*, *Z*, *C*
Logical	`ANDS, ORRS, EORS, BICS`	*N*, *Z*, *C*
Test	`TEQ, TST`	*N*, *Z*, *C*
Move	`MOVS, MVNS`	*N*, *Z*, *C*
Multiply	`MULS, MLAS, SMLALS, SMULLS, UMLALS, UMULLS`	*N*, *Z*

Table 6.3 Condition mnemonics

cond	Mnemonic	Name	CondEx
0000	EQ	Equal	Z
0001	NE	Not equal	$\overline{Z}$
0010	CS/HS	Carry set / unsigned higher or same	C
0011	CC/LO	Carry clear / unsigned lower	$\overline{C}$
0100	MI	Minus / negative	N
0101	PL	Plus / positive or zero	$\overline{N}$
0110	VS	Overflow / overflow set	V
0111	VC	No overflow / overflow clear	$\overline{V}$
1000	HI	Unsigned higher	$\overline{Z}C$
1001	LS	Unsigned lower or same	Z OR $\overline{C}$
1010	GE	Signed greater than or equal	$\overline{N \oplus V}$
1011	LT	Signed less than	$N \oplus V$
1100	GT	Signed greater than	$\overline{Z}(\overline{N \oplus V})$
1101	LE	Signed less than or equal	Z OR $(N \oplus V)$
1110	AL (or none)	Always / unconditional	Ignored

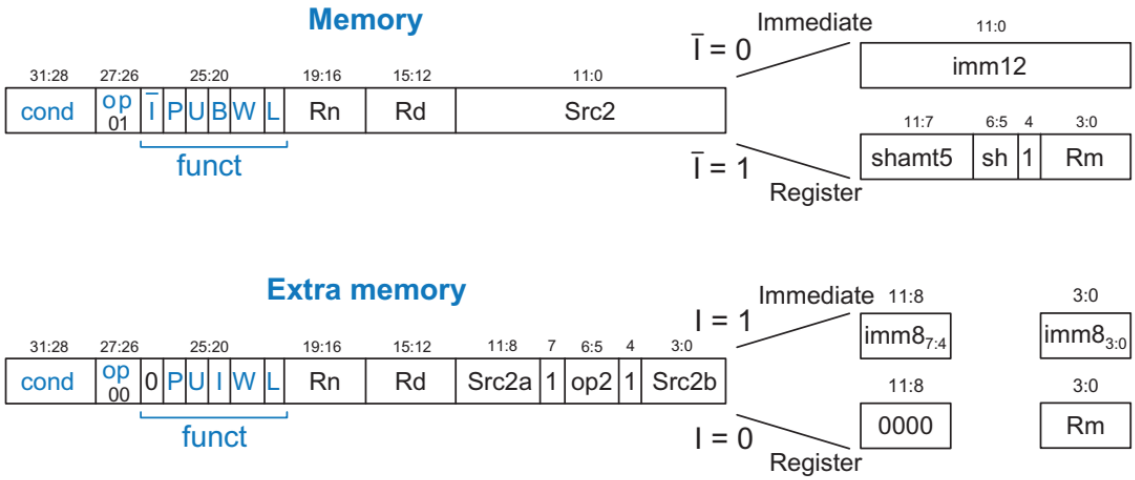


Figure B.3 Memory instruction encodings

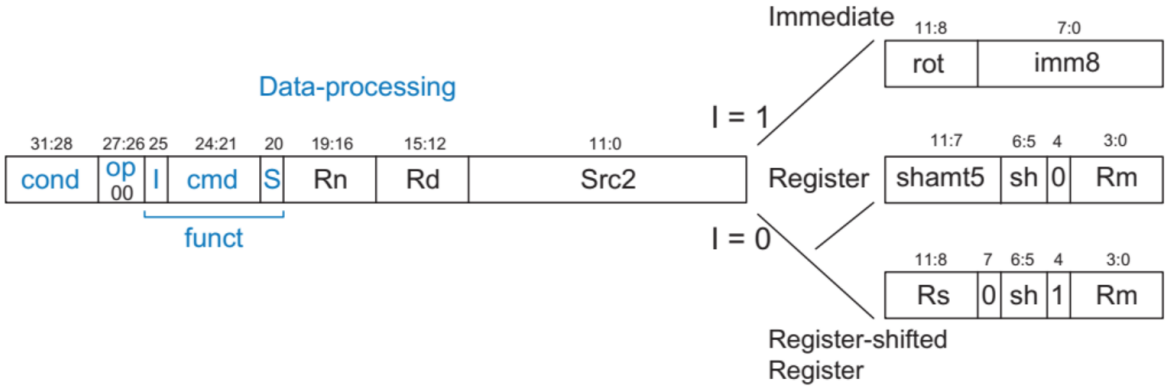


Figure B.1 Data-processing instruction encodings

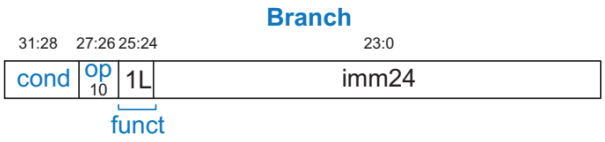


Figure B.4 Branch instruction encoding

Table B.4 Branch instructions

L	Name	Description	Operation
0	`B label`	Branch	`PC ← (PC+8)+imm24 << 2`
1	`BL label`	Branch with Link	`LR ← (PC+8) - 4; PC ← (PC+8)+imm24 << 2`

Table B.1 Data-processing instructions

cmd	Name	Description	Operation
0000	`AND Rd, Rn, Src2`	Bitwise AND	`Rd ← Rn & Src2`
0001	`EOR Rd, Rn, Src2`	Bitwise XOR	`Rd ← Rn ^ Src2`
0010	`SUB Rd, Rn, Src2`	Subtract	`Rd ← Rn - Src2`
0011	`RSB Rd, Rn, Src2`	Reverse Subtract	`Rd ← Src2 - Rn`
0100	`ADD Rd, Rn, Src2`	Add	`Rd ← Rn+Src2`
0101	`ADC Rd, Rn, Src2`	Add with Carry	`Rd ← Rn+Src2+C`
0110	`SBC Rd, Rn, Src2`	Subtract with Carry	`Rd ← Rn - Src2 -` $\overline{\texttt{C}}$
0111	`RSC Rd, Rn, Src2`	Reverse Sub w/ Carry	`Rd ← Src2 - Rn -` $\overline{\texttt{C}}$
1000 ($S = 1$)	`TST Rd, Rn, Src2`	Test	`Set flags based on Rn & Src2`
1001 ($S = 1$)	`TEQ Rd, Rn, Src2`	Test Equivalence	`Set flags based on Rn ^ Src2`
1010 ($S = 1$)	`CMP Rn, Src2`	Compare	`Set flags based on Rn - Src2`
1011 ($S = 1$)	`CMN Rn, Src2`	Compare Negative	`Set flags based on Rn+Src2`
1100	`ORR Rd, Rn, Src2`	Bitwise OR	`Rd ← Rn | Src2`
1101	Shifts:		
$I = 1$ OR ($\text{instr}_{11:4} = 0$)	`MOV Rd, Src2`	Move	`Rd ← Src2`
$I = 0$ AND ($sh = 00$; $\text{instr}_{11:4} \neq 0$)	`LSL Rd, Rm, Rs/shamt5`	Logical Shift Left	`Rd ← Rm << Src2`
$I = 0$ AND ($sh = 01$)	`LSR Rd, Rm, Rs/shamt5`	Logical Shift Right	`Rd ← Rm >> Src2`

Scratch paper

Scratch paper