

Practice Midterm

1

Replace the following two instructions with one:

```
MOVE.L    A0, A1
ADDA.W    D3, A1
```

2

Suppose you working on a new program for the GameGirl hand-held video game machine, powered by a Motorola 68k CPU and a co-worker comes to you with the following code segment. This person tells you that this fragment is intended to store the value \$01FE into the first element of an array of words and then jump to a system routine. Explain at least two problems you see with this code. Assume that the fragment is contained in a program loaded between address \$1000 and address \$2302, and that D0 = \$00000000.

```
Routine    EQU        $3203
TABLE      DC.W        100

Stor       PROC
           MOVE.W     #$01FE, TABLE(PC, D0.L)
           JMP        Routine
           ...
```

3

The following are machine code instructions including opcode and extension word. Show the corresponding instruction. If this is an invalid instruction, please note it.

| Machine Code | Hint | Specific Instruction |
|----------------|------|----------------------|
| 48C4 | EXT | |
| 4470 18FC | NEG | |
| 11FA FFED 3933 | MOVE | |
| DBA8 0028 | ADD | |

4

For each of the instructions in the fragment of code below, fill in the blank spaces with the instruction encoding. Assume the following conditions:

- The program is loaded so that the first instruction (MOVEQ) is loaded at address \$2000
- thing is located -\$4 from A5

| Instruction | Machine Code |
|------------------------|--------------|
| stuff DC.W \$F,\$2,\$B | |
| doo DC.L 12 | |
| da EQU12 | |
| thing DS.L 10 | |
| MOVEQ #-9, D2 | |
| EORI.W #da, thing(A5) | |
| ADD.B doo(PC,D0.L), D7 | |
| LEAda, A3 | |

5

For the following code, give the number of times memory is accessed to fetch the instructions, to read the operands, and to write the operands. Assume 16 bits can be transferred at once (you have a 16-bit bus). Hint: JSR needs to push the return address on the stack before jumping to the subroutine). The following code precedes the instructions:

```

ROMroutine EQU $0050FF42
ParamOne EQU $8
Vars DS.B 6

Constants DC.W $3000, $4000, $5000, $6000
...
JuggleBits PROC ...

```

| Instruction | Fetch Reads | Execute Reads | Execute Writes |
|----------------------------------|-------------|---------------|----------------|
| JSR ROMroutine.L | | | |
| LEA Vars(A5), A0 | | | |
| MOVE.W (A7)+, ParamOne(A1, D0.L) | | | |
| CMP.L \$22, (A1) | | | |
| JSR JuggleBits(PC) | | | |
| ADD.W Constants(PC, D0.W), D2 | | | |

6

Given the following conditions, for each of the instructions below, note in hex the destination effective address and the value placed in memory after the execution of the instruction. Assume that every instruction is executed with the following initial conditions:

| | |
|---------------------------|------|
| 0000 3300 | FA22 |
| 0000 3302 | 2044 |
| 0000 3304 | 0001 |
| 0000 3306 | FFF8 |
| 0000 3308 | 2206 |
| 0000 330A | 11C0 |
| 0000 330C | 9E72 |
| 0000 330E | 0204 |
| 0000 3310 | C00F |
| 0000 3312 | CAB0 |
| 0000 3314 | B00F |
| 0000 3316 | D00F |
| | |
| 0000 3318 to FFFF FFFF | 0000 |

| | |
|----|-----------|
| A0 | 0000 3300 |
| A1 | 0000 3302 |
| A2 | 0000 200A |
| A3 | BEEF BBAD |
| A4 | EEAA 7EEE |
| A5 | 0005 3002 |
| A6 | FFFF F380 |
| A7 | 0000 3304 |
| D0 | 0870 FFFC |
| D1 | 0000 0002 |
| | |
| PC | 0000 1000 |

vals DC.B 22, -3, -2, -1
 scrud DS.L 14
 scsi EQU \$7FF4
 lpr EQU \$7FF8

| | | Destination Effective Address in hex | 32-bit value in hex at destination address after instruction executes |
|--------|--------------------|--|---|
| MOVE.B | vals + 2(PC), lpr | | |
| MOVE.L | vals(PC), \$1C(A1) | | |
| MOVE.L | #lpr, 20(A0, D0.W) | | |
| MOVE.W | \$3314, 4(A1) | | |
| MOVE.W | vals(PC,D1), -(A7) | | |

7

Write the state of the condition codes and the contents of D0 after each of the following instructions (assume that they execute continuously, in the order given).

| Instruction | D0 | N | Z | V | C |
|------------------------|-----------|---|---|---|---|
| MOVE.W #\$801C, D0 | 1F2B 801C | 1 | 0 | 0 | 0 |

| | | | | | |
|--------|-----------------|--|--|--|--|
| SUBI.W | #\$19FF, D0 | | | | |
| ADDI.W | #\$7F7F, D0 | | | | |
| BTST | #6, D0 | | | | |
| EORI.L | #\$F0F0F0F0, D0 | | | | |

8

Suppose you have the following case table in a game you're writing in a high level language:

```

switch (ch) {
    case '8': Up(Spaceship); break;
    case '2': Down(Spaceship); break;
    case '4': Left(Spaceship); break;
    case '6': Right(Spaceship); break;
    case '0': Fire(Spaceship); break;
}

```

where Down, Left, Right, Up, and Fire are routines you've written to handle those commands. Here is a table of the ASCII values for those characters:

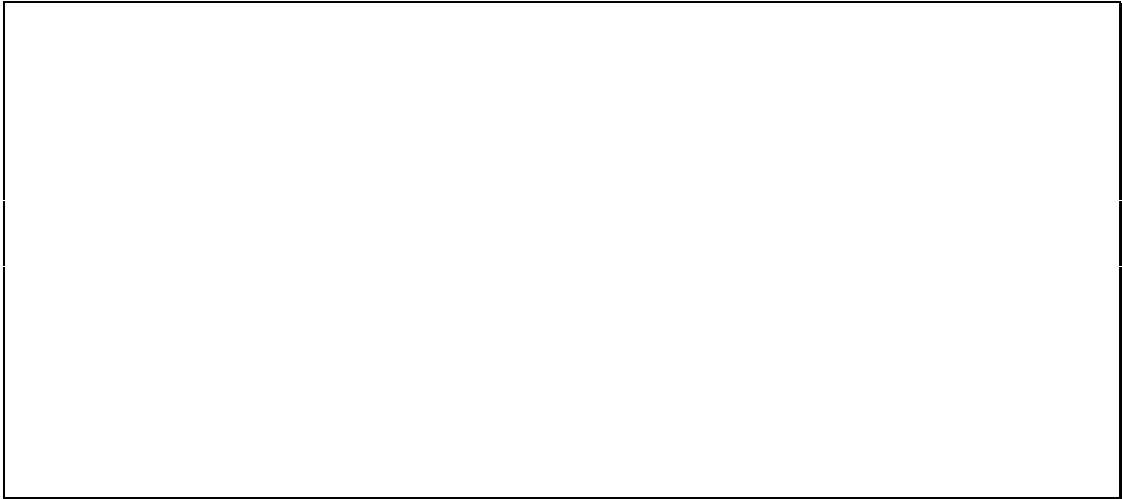
| Character | Decimal | Hex | Meaning |
|-----------|---------|-----|---------|
| '0' | 48 | 30 | Fire |
| '2' | 50 | 32 | Down |
| '4' | 52 | 34 | Left |
| '6' | 54 | 36 | Right |
| '8' | 56 | 38 | Up |

Create an efficient jump table that handles your game's input. A code skeleton is provided below. Assume that the input has already been filtered for you, i.e., that the only values that you will be passed are the five characters listed above. You should not need more than the space provided. Also show the code needed to access the jump table:

```

Char DS.B 1 ; User input key
...
HandleChar MOVE.B Char(A0), D0 ; Put user input in D0
...

```



```
    ...  
Up   ADDI  #1, Vert_Loc(A5)    ; Move spaceship up  
    ...  
Down SUBI  #1, Vert_Loc(A5)   ; Move spaceship down  
    ...  
Left  SUBI  #1, Horiz_Loc(A5) ; Move spaceship left  
    ...  
Right ADDI  #1, Horiz_Loc(A5) ; Move spaceship right  
    ...  
Fire  SUBI  #1, Num_Shots(A5) ; Cost one bullet
```