ECE 331 Homework 2

Handout

ECE 331 – Fall 2024
Homework 2
Assigned Monday, September 16th
Due Monday, September 23rd

1. Hex representations of three Arm instructions are given below: (10)

(a) 0x8b14020b

(b) 0xf8460198

(c) 0xd1155869

For each of these instructions, answer/do the following:

• Write the 32-bit binary representation of this instruction.

• What instruction is this (name and mnemonic)?

• What type (R, I, D) is this instruction?

• Specify the contents of each field.

Express opcode in hexadecimal and all other fields in decimal.

• Write this instruction in Arm assembly language.

• In words, what does this instruction do? Be concise and precise.

2. (a) Explain the need for subroutines. (5)

(b) Explain the steps involved in a procedure call. (5)

3. Consider the following code: (10)

```
if(i == j){f=g+h; f=f+1;}
else f=f+1;
```

Write a sequence of ARM assembly instructions that will execute this line of code...

(a) ... assuming that the values of f, g and h are stored in x19, x20 and x21, respectively. The values i and j are stored in x9 and x10, respectively.

(b) ... assuming instead that the values of f, g and h are stored at memory locations 68100, 68108, 68116, respectively, and the values i and j are stored

Work

1. Hex representations of three Arm instructions are given below:
    1. (a) 0×8b14020b
    2. (b) 0xf8460198
    3. (c) 0xd1155869
    - For each of these instructions, answer/do the following:
    - Write the 32-bit binary representation of this instruction.
        1. 10001011000101000000001000001011
        2. 11111000010001100000000110011000
        3. 11010001000101010101100001101001
    - What instruction is this (name and mnemonic)?
        1. ADD, add
        2. LDUR, Load Register Unscaled Offset
        3. SUBI, Subtract Immediate
    - What type (R, I, D) is this instruction?
        1. R
        2. D
        3. I
    - Specify the contents of each field. Express opcode in hexadecimal and all other fields in decimal.
        1. opcode 458, Rm 20, shamt 0, Rn 16, Rd 11
        2. opcode 7C2, DT address 192, op 0, Rd 12, Rt 24
        3. opcode 688-689, Rd 9, Rn 3, ALU Immediate 683
    - Write this instruction in Arm assembly language.
        1. ADD x11, x16, x20
        2. LDUR x24, [x12, 192]
        3. SUBI x3, x9, #683
    - In words, what does this instruction do? Be concise and precise.
        1. add whats in x16 and x20 and store in x11
        2. load whats in register 12 #192 to x24
        3. subtract 683 from x9 and store in x3
2. (a) Explain the need for subroutines.
    1. subroutines are also known as functions give much more flexibility to your code, it allows you to use those parts of code in other places to save time, it makes each part of the code responsible for one function which allows

easier debugging, it also allows for easier coding by breaking down difficult problems into smaller easier to tackle parts

2. (b) Explain the steps involved in a procedure call. (got straight from slides)
    1. The calling program transfers control to the procedure (callee)
        1. enters the subroutine
    2. The called procedure acquires storage that it needs from memory
        1. makes space in memory to hold whatever information it needs to use in the operations
    3. The called procedure executes its operations
        1. applies operations to the data that is stored
    4. The called procedure places results in registers for the calling program to retrieve.
        1. the outputs from the operations are stored in the allocated memory spots
    5. The called procedure reverts the appropriate ARM registers to their original or correct state.
        1. the registers such as zero negative or overflow registers are put back to how they were before to not affect other functions
    6. The called procedure returns control to the the next word in memory from which it was called.
        1. sends position back to where it was before, incrementing by 1 step to the next instruction
    7. The calling program proceeds with its calculation
        1. business resumes as normal

3. Consider the following code: `if(i == j){f=g+h; f=f+1;} else f=f+1`; Write a sequence of ARM assembly instructions that will execute this line of code...

    - (a) ... assuming that the values of f, g and h are stored in x19, x20 and x21, respectively. The values i and j are stored in x9 and x10, respectively.

```
Fact:
        SUB XZR, x9, x10 #compare x9 and x10
        BNE Else #if different, jump to else
subroutine
        ADD x19,x20,x21 #perform f=g+h
        BR LR #return
Else:
        ADDI x19,x19,#1 #perform f=f+1
        BR LR #return
```

- (b) ... assuming instead that the values of f, g and h are stored at memory locations 68100, 68108, 68116, respectively, and the values i and j are stored in x9 and x10, respectively. Assume also that the decimal value 68092 resides in register x19.

```
                LDUR x22,[x19,#0]
                LDUR x20,[x19,#8]
                LDUR X21,[x19,#16]
    Fact:
                SUB XZR, x9, x10
                BNE Else
                ADD x22,x20,x21
                STUR x22,[x19,#0]
                BR LR
    Else:
                ADDI x22,x22,#1
                STUR x22,[x19,#0]
                BR LR
```

4. Logical Operations:

- (a) For what practical applications in a program can logical operations be Used?

    - as is said on the slide, logical operations are good for inserting and extracting a group of bits in a word, this can be applied to embedded systems where you need to change specific bits of the register so you utilize bit mapping to change just the ones you need
- (b) Write ARM code (a few instructions) demonstrating the applications you listed in part (a).
- say you want to change specific bits of a register to change for example; from output to input, where you want to change 11111111 to 11110000 in register x19 68108 to make the last 4 pins output instead of input. the best way to do this is to use a logical shift left

```
        LDUR x11,[x19,#8]  #load register x19 68108 in
x11 (11111111)
        LSL x11, x11, 4    #perform left shift to get
11110000
        STUR x11, [x19,#8] #store back in stack
```

5. Consider the following C code: (10)

```c
unsigned long long main()
{
unsigned long long x = 1;
switch (x)
{
case 1: i=1;
case 2: i=5;
}
return 0;
}
```

5.
- Based on this code, answer the following:
- (a) Convert the above C code to ARM assembly. Assume that unsigned long long corresponds to a 64-bit unsigned integer datatype. (lets say I is

in register x2)

```
Fact:
        ADDI x1,XZR, #1 #assign x to 1
        BR Switch #move to next function
Switch:
        SUBS XZR,x1,#1 #check if x is equal to 1
        BEQ Case1 #jump to x=1 subroutine
        SUBS XZR,x1,#2 #check if x is equal to 2
        BEQ Case2 #jump to x=2 subroutine
        B else #if not either, jump to else case
Case1:
        ADDI x2, XZR, #1 #assign i to 1
        B LR #return
Case2:
        ADDI x2, XZR, #5 #assign i to 5
        B LR #return
else:
        ADDI x2, XZR, XZR #assign i to 0
        B LR #return
```

- (b) The condition you are testing is met at Case 1. You do not want the Case 2 to be tested. Add appropriate instructions in Arm for this implementation.
- (c) If int x is any integer other than 1 or 2, you want ‘i’ to be 0. First, add C code that implements this, then add the appropriate instructions in ARM for this implementation.

6. . a) Convert the following C code to Arm: (10)

```c
unsigned long long sum( unsigned long long arr[], unsigned
long long size ) {
if ( size == 0 )
return 0 ;
else
return sum( arr, size - 1 ) + arr[ size - 1 ] ;
}
```

6. Assume arr is in x19 and size is in x20 and follow all standard ARM conventions for procedures. Also, assume that the unsigned long long data type corresponds to a 64-bit unsigned integer.

```
Fact:
    SUBS    x0, x20, #0           # if size == 0
    B.EQ    else                  # if equal, jump to else
    STP     x29, x30, [sp, #-16]! # save frame pointer
(x29) and link register (x30)
    MOV     x29, sp               # update frame pointer
    STP     x19, x20, [sp, #-16]! # save arr (x19) and size
(x20)

    SUB     x20, x20, #1          # size - 1
    BL      sum                   # recursive call:
sum(arr, size - 1)

    LDP     x19, x20, [sp], #16   # restore arr (x19) and
size (x20)
    ADD     x0, x0, [x19, x20, LSL #3] # add arr[size - 1]
to the result (x0)

    LDP     x29, x30, [sp], #16   # restore frame pointer
and link register
    RET                           # return result

else:
    MOV     x0, #0                # return 0 for base case
    RET                           # return
```

- b) What else would be saved on the stack?
  - the link register (x30)
  - frame pointer (x29)
  - saved registers (x19, x20)