ELEC6027: VLSI Design Project
Part 1: Microprocessor Research
Topic: Subroutines

Ashley Robinson
Team: R4
Course Tutor: Mr B. Iain McNally

11$^{\text{th}}$ February, 2014

Contents

1 Introduction

Subroutines, also known as procedures, methods, functions or just routines, are smaller sections of code inside larger program designed to perform certain tasks [2]. The motivation for subroutines is to produce code which is more efficient in size, easy to adapt and above all else maintain. They help form the foundations of third generation programming languages.

Designing hardware such that it is capable of executing subroutines only requires available memory and access to the program counter. Designing hardware to call and return from subroutines efficiently can vastly improve the performance of a processor.

2 Research

2.1 Subroutine Context Save

Context save, also known as, allows

2.2 Operation of Stack Frames

2.2.1 8086

The assembler held in listing 1 and 2 is written for the Intel 8086 microprocessor. A basic example of how stack frames are built to pass parameters to and from a subroutine. The main program in listing 1 loads two immediate values into registers then begins building a stack frame by pushing them to the stack. The subroutine is called to act upon the arguments passed via the stack. When control is passed back to these set of instructions and the return value is extracted by using relative addressing from the base pointer then finally two stack pops completely destroy the stack frame.

```
main:                       ; Main loop
        MOV     bp,sp       ; Init base ptr
        MOV     ax,42       ; Load arg1
        MOV     bx,69       ; Load arg2
        PUSH    bx          ; Push arg1 to stack
        PUSH    ax          ; Push arg2 to stack
        CALL    adder       ; Call the subroutine
        MOV     cx,[bp-12]  ; Access return value
        POP     ax          ; Restore all registers
        POP     bx
        JMP     main
```

Listing 1: 8086Caller.asm

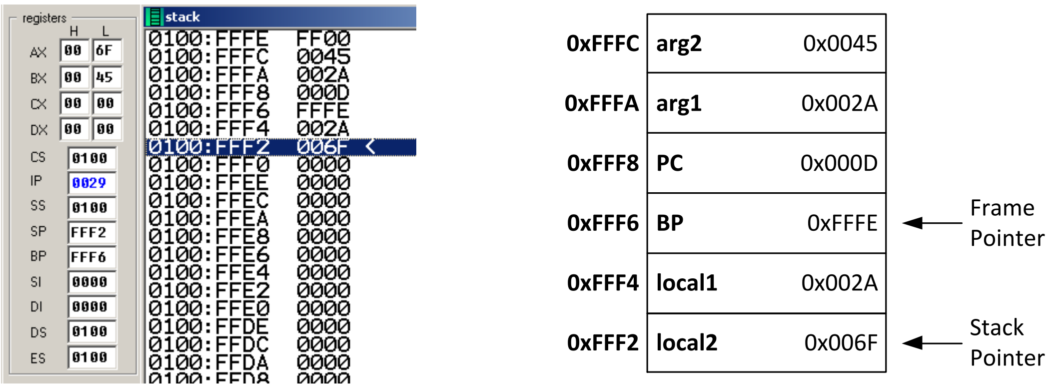
When the subroutine, in listing 2, is called the return address is pushed onto the stack. This built-in support for the stack handles branching and next line address storage using a call function. To start the base pointer is placed on the stack so stack pointer has value to which to be restored. Reducing the value of the stack pointer allocates space for local variables. The first argument is placed in memory as local variable; this is unnecessary but serves as example. The second argument is loaded into a working register. The first local variable is added to the working register which is then placed in to the memory for the second local variable. Finally the stack pointer and the base pointer are restored and a return instruction hands control over the caller. This is all part of the calling convention for subroutines using stack frames on the 8086 [1].

```
adder PROC                  ; Subroutine
        PUSH    bp          ; Push base ptr to stack
        MOV     bp,sp       ; Set base ptr to stack ptr
        SUB     sp,4        ; Allocate local variable space (2 ints)
        MOV     ax,[bp+4]   ; Load arg1 into Working
        MOV     [bp-2],ax   ; Load arg1 into Local1
        MOV     ax,[bp+6]   ; Load arg2 into Working
        ADD     ax,[bp-2]   ; Add to contents of working reg
        MOV     [bp-4],ax   ; Write Local2 with result
```

```
        MOV     sp,bp           ; Return stack ptr
        POP     bp              ; Restore base ptr
        RET                     ; Done
adder ENDP
```

Listing 2: 8086Callee.asm

This code was tested upon an 8086 emulator [5]. The emulator provides a complete overview of the flow of data within the processor including the stack. Figure 1a shows the emulator during the execution of the subroutine just before the stack pointer is overwritten with the base pointer. Figure 1b is an abstraction of the stack with data label corresponding to the subroutine.



(a) Emulator registers and stack

(b) Stack frame configuration

Figure 1: 8086 stack operation

2.2.2 ARM7TDMI using Arm Thumb

The ARM7TDMI is a 32-bit RISC microprocessor with an emphasis on low-power design and pipelining for high throughput [3]. It has two instruction sets one of which is Arm Thumb, a low density 16-bit subset of the ARM assembly language [4]. A user selectable flag is set to switch between instruction sets therefore drawing on each sets advantages.

This architecture does not have built-in support for calling subroutines using the stack. When the branch instruction is used, as seen in listing 3, the program counter is overwritten with the address of the corresponding label. The address of the next line of code, which should be returned to after the subroutine, is placed into the link register. Calling conventions suggests leaving this register untouched and simply moving the data back into the program counter on a return.

```
main  MOV   r0,#42    ; Load arg1
      MOV   r1,#69    ; Load arg2
      PUSH  r0        ; Push arg1 to stack
      PUSH  r1        ; Push arg2 to stack
      BL    adder     ; Branch to subroutine
      POP   r0        ; This line is held in the link register
      POP   r0        ; Result pop from arg1 spot
      BL    main
```

Listing 3: ArmCaller.asm

In this case the link register is pushed onto the stack from the subroutine therefore requiring the subroutine to pop the value into the program counter in order to return. Listing 4 holds the subroutine and handles placing the return address on the stack. Relative addressing on the stack is required to draw the two arguments out and replace the first with the output of the function.

```
adder PUSH  lr               ; Link register holds return address
      LDR   r0, [sp,#12]     ; Get arg1 off stack
      LDR   r1, [sp,#8]      ; Get arg2 off stack
      ADD   r0,r1            ; Do the add
      STR   [sp,#12], r0     ; Replace arg1 on the stack
      POP   pc               ; Restore program counter and return
```

Listing 4: ArmCallee.asm

3 Conclusion

References

[1] James Archibald. The c calling convention and the 8086: Using the stack frame. `http://ece425web.groups.et.byu.net/stable/labs/StackFrame.html`, 2013. Online. Acessed Feb 2014.

[2] Allison June Barlow Chaney. Subroutine. `https://www.princeton.edu/~achaney/tmve/wiki100k/docs/Subroutine.html`, Oct 2013. Online. Acessed Feb 2014.

[3] ARM Holdings plc. Arm7tdmi data sheet. `http://www.ndsretro.com/download/ARM7TDMI.pdf`, Aug 1995. Online. Acessed Feb 2014.

[4] ARM Holdings plc. Thumb instruction set quick reference card. `http://www.eng.auburn.edu/~nelson/courses/elec5260_6260/Thumb%20Instructions.pdf`, Oct 2003. Online. Acessed Feb 2014.

[5] Daniel B. Sedory, Randall Hyde, Eric Isaacson, Barry Allyn, Tomasz Grysztar, Saul Coval, Bob Brodt, Jordan Russell, and Jeremy Gordonii. emu8086. `http://www.emu8086.com/`, 2013. Online. Acessed Feb 2014.

Bibliography

[1] Patterson D A and Hennessy J L, *Computer Organisation and Design: The Hardware/Software Interface*. Morgan Kaufman, 4th Edition, 2009. - Lots of processor concepts and MIPS examples