# 1 April 2002

## Subroutines

`X = sqrt(Y) ;`

How do you get to `sqrt`?
How do you return to caller?
How do you pass the argument `Y`?
How do you retrieve the returned value?

## Entering the subroutine in LC-2

Requires an instruction to branch and remember the return

`JSR` – jumps to subroutine and saves PC in R7

```
            JSR     sqrt
```

`JSRR` – jumps to subroutine via register and saves PC in R7

```
            LD      R5,Asqrt
            JSRR    R5,#0
            .....
Asqrt       .FILL   sqrt
```

`JSR/JMP` and `JSRR/JMPR` share the same opcode

If bit 11 is 1, the subroutine jumps are used

## Returning from the subroutine in LC-2

Branch to value saved in R7

## Calling conventions

Where are arguments located?
How are values returned?
What registers can be used without saving?

## Passing arguments in LC-2

- Could pass them in registers, if not too many
  - Alpha uses R16 to R21
- Could use an *activation record* – more in chapters 10 and 14
  - Intel x86 uses a stack

## Activation format in the LC-2

R6 points to bottom (*lowest address*) of activation record

| | |
|---|---|
| Return value | Stored by callee |
| Return address | Saved in R7 by caller, stored by callee |
| Dynamic link | *Wait until chapter 14* |
| Passed arguments | Stored by caller |
| | |
| Local variables | *Wait until chapter 14* |
| | |

## Returning values in the LC-2

Registers (R0)

## Example 1

Function to multiply R0 by 10

## Example 2

Write function `prntint`

- Single integer argument passed in R0
- Writes R0 in decimal to the CRT
- Returns in R0 the number of characters printed

Using function `div10`

- Receives a dividend in R0
- Returns quotient in R1
- Returns remainder in R2

## Stacks

For storing temporary values in expression evaluation

```
x = (a+b) * (c+d) ;
```

For storing activation records for recursive computations

```
int fact(int x) {
    if (x<=1)
        return 1 ;
    else
        return x*fact(x-1) ;
}
```

## History

*Polish notation*

Invented by Jan Lukasiewicz (1878-1956)

Eliminates parentheses in mathematical expressions

(a+b) * (c+d) ⇒ * + a b + c d

Also, called *prefix* notation

```
times(plus(a,b),plus(c,d))
(times (plus a b) (plus c d))
```

Reverse Polish notation (RPN)

Puts the operators after the operands

(a+b) * (c+d) ⇒ a b + c d + *

## Computational examples of RPN

Popularized in HP RPN calculators

Operands held in a *stack*

17 `[enter]` 23 `[enter]` + 43 `[enter]` 57 `[enter]` + *

Widely done in Java

CoCalc RPN Scientific Calculator

Stack computers

Intel x86 stack pointer stored in register `SP` (or `ESP`)

SP grows downward in memory

Push bytes, words, double words, all registers

Primary use of stack is to store activation records

Stack overflow is a serious Internet security problem!

Stack based programming languages

Forth

Invented by Charles Moore in early 60's

Used to control many real-time devices (FedEx wand)

Open Firmware standard used to boot devices

Postscript

Invented by Adobe

Used in many laser printers

How is this document generated?

MS Word ⇒ PostScript ⇒ PDF

How about the trees at the right?

Pure PostScript

## Stack on the LC-2

R6 points to top value on the stack

It's a little messy when the stack is empty

LC-2 stack grows upward

Most stacks grow downward

Simple PUSH operation

```
PUSH    ADD     R6,R6,#1
        STR     R0,R6,#0
        RET
```

Simple POP operation

```
POP     LDR     R6,R6,#0
        ADD     R6,R6,#-1
        RET
```

Robust implementations check for stack underflow and overflow

Special arithmetic operations can use the stack

```
OpAdd   ADD     R6,R6,#-1
        LDR     R0,R6,#0
        LDR     R1,R6,#1
        ADD     R0,R0,R1
        ST      R0,R6,#0
        RET
```

Try an example (or two)

```
x = (a+b) * (c+d) ;
x = a*(b+c) + d*(b-c) ;
```

# Interrupt handling

New Section 10.5

What's wrong with this code?

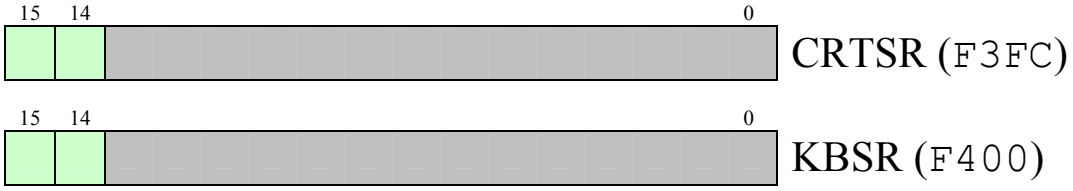
```
        ; Loop waiting for a new character
INLOOP  LDI     R0,AKBSR
        BRzp    INLOOP
        ; Read the new character
        LDI     R0,AKBDR
```

Computer executes 100,000,000 instructions per second
Person types 10 characters per second
    99.99999% of the time is spent "spinning"

## Tools for interrupts

- Interrupt signal
  - Bit that tells the CPU if interrupt has occurred
- Means of saving "state" of running program
- Means of restoring "state" of interrupted program
- Means of disabling and enabling interrupts
- Means of determining interrupting device

## LC/2 Interrupt Enable bit




Bit 15 – device ready bit
Bit 14 – device interrupt enable bit

If both bits are set for a device, interrupt the next instruction!

Initiating the interrupt

- Save state of executing program
  - Store the PC — M[R6+1] ← PC
  - Store the N,Z,P bits — M[R6+2] ← N,Z,P
  - Increase "stack" — R6 ← R6 + 2
- Transfer to interrupt handler
  - Load PC with address provided by the device
    - This is highly non-standard

Executing the interrupt handler

- Interrupt handler performs I/O operation
  - And may initiate a new operation
- On real computers
  - Handler may "mask" additional interrupts
- On completion, handler returns with `RTI` instruction
  - Decrease "stack" — R6 ← R6 - 2
  - Restore the PC — PC ← M[R6+1]
  - Restore the N,Z,P bits — N,Z,P ← M[R6+2]

An example

- "Main" loop increments COUNT1 continuous
- Interrupt handler increment COUNT2 on character input

Version A

- Has problems if interrupt occurs when "main" increments

Version B

- Has a problem with two quick interrupts

A real computer needs

- An **atomic** means of entering and exiting interrupt handlers

Download all lecture ASM files in ZIP format