Subroutines

Subroutines

Definition

A **subroutine** is a sequence of instructions that can be called as a single unit as part of a larger program.

- Subroutines are also called **procedures**.
  - All functions are subroutines.
  - Not all subroutines are (pure) functions.
- Subroutines allow code for common tasks to be reused.

Example

```
1 ; Computes R2 = R0 - R1.
2 NOT R3, R1        ; Negate R1.
3 ADD R3, R3, #1
4 ADD R2, R0, R3    ; Add -R1 to R0.
```

Subroutines

- The first address in a subroutine is constant.
- The **return address** in its caller is variable.

Example *(cont.)*

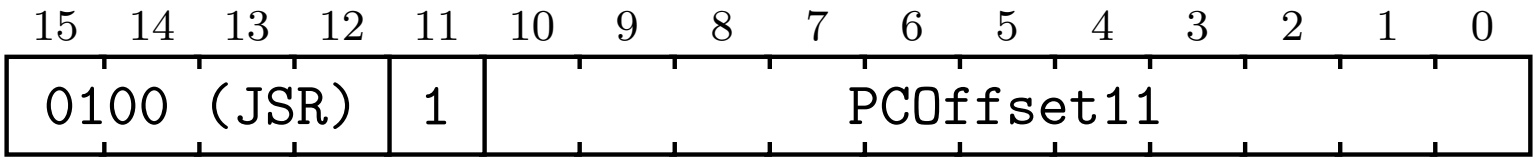
Consider the following "subroutine":

```
          BRnzp SUB          ; "Call" SUB.
DONE      ; Do something with R2.


; Computes R2 = R0 - R1.
SUB       NOT R3, R1         ; Negate R1.
          ADD R3, R3, #1
          ADD R2, R0, R3     ; Add -R1 to R0.
          BRnzp DONE         ; "Return" to caller.
```

This cannot be reused; it would always return to DONE.

JSR (Jump to Subroutine, PC-Relative)



1. Sign-extend `PCOffset11` to 16 bits and add it to the PC.
2. Place that address into the PC.
3. Place the old value of the PC into `R7`.

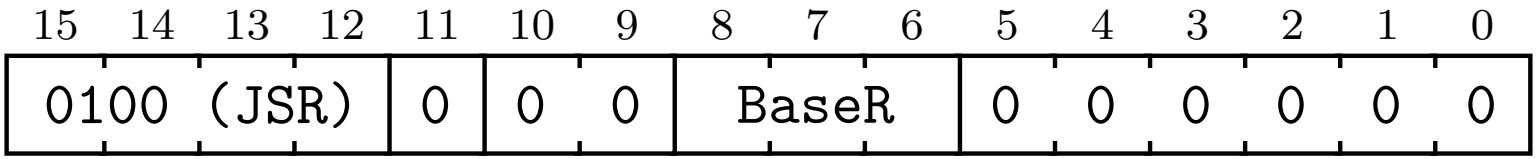
Example

Suppose the PC contains `0x3001`. Then executing:

`0100 1 01100000000`

...results in the PC's containing `0x3302`, and `R7`, `0x3002`.

JSRR (Jump to Subroutine, Register-Based)



1. Place the value of BaseR into the PC.
2. Place the old value of the PC into R7.

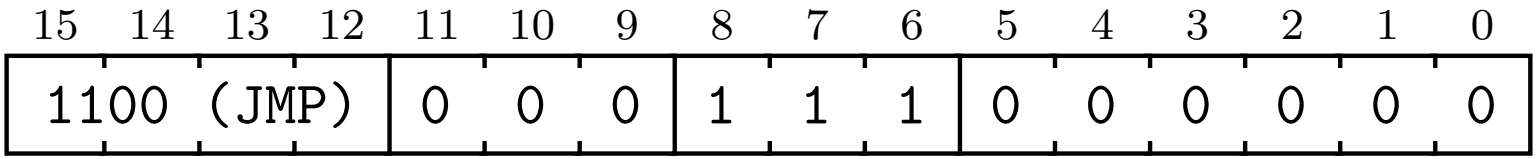
Example

Suppose the PC contains 0x3001 and R4 contains 0x3302. Then executing:

0100 000 100 000000

...results in the PC's containing 0x3302, and R7, 0x3002.

RET (Return from Subroutine)



1. Place the old value of R7 into the PC.

Example

Suppose R7 contains 0x3002. Then executing:

```
1100 000 111 000000
```

...results in the PC's containing 0x3002.

Arguments and Return Values

- Arguments and return values may be passed via registers.
  - The hardware has no requirements for which registers are used.
  - The assembler will not check that the correct types or numbers of values have been passed.
- This convention is limited to 7 arguments and return values.

Example *(cont.)*

```
1 ; Subtracts one integer from another.
2 ;  Takes the minuend in R0, subtrahend in R1.
3 ;  Returns the difference in R2.
4 SUB     NOT R3, R1       ; Negate R1.
5         ADD R3, R3, #1
6         ADD R2, R0, R3   ; Add -R1 to R0.
7         RET
```

Saving and Restoring Registers

- Registers altered by a subroutine must be saved and restored.

Definition

The calling program is responsible for saving **caller-save** registers.

- Registers used for return values along with `R7` are caller-save.
- The caller is not obligated to save registers it doesn't need.

Definition

The called subroutine is responsible for saving **callee-save** registers.

- Any other registers used by the subroutine are callee-save.
- The callee *must* save these registers.

Saving and Restoring Registers

Example *(cont.)*

```
1  ; Subtracts one integer from another.
2  ;  Takes the minuend in R0, subtrahend in R1.
3  ;  Returns the difference in R2.
4
5  SUB     ST  R3, SAVER3  ; Save R3.
6          NOT R3, R1      ; Negate R1.
7          ADD R3, R3, #1
8          ADD R2, R0, R3  ; Add -R1 to R0.
9          LD  R3, SAVER3  ; Restore R3.
10         RET
11
12 SAVER3  .FILL x0000     ; Space to save R3
```

Calling Subroutines

In order to call a subroutine, the programmer must know:

- Where to place its expected arguments
- Where to find its eventual return values
- Its starting address

Example *(cont.)*

```
                ; Place arguments into R0 and R1
1               ST  R7, SAVER7  ; Save R7.
2               JSR SUB         ; Call subroutine.
3               LD  R7, SAVER7  ; Restore R7.
                ; Result has been placed into R2.

SAVER7  .FILL 0x0000    ; Space to save R7
```

Nested Subroutine Calls

Example

```
1  ||          .ORIG x3000
2  | MAIN      JSR FOO
3  |           HALT
4  |
5  | FOO       ST  R7, SAVER7   ; The call to BAR will
6  |           JSR BAR          ;  overwrite the return
7  |           LD  R7, SAVER7   ;  address in R7.
8  |           RET
9  | SAVER7    .FILL x0000
10 |
11 | BAR       RET
12 ||          .END
```

- MAIN does not need to save R7.
- FOO needs to save R7, else it would infinitely loop on return.

Recursive Subroutine Calls

Example

```
1  |          .ORIG x3000
2  | MAIN     JSR FOO
3  |          HALT
4  |
5  | FOO      ADD R0, R0, #-1 ; Decrement R0.
6  |          BRnz DONE       ; If R0 is positive...
7  |          ST  R7, SAVER7  ; ...recurse...
8  |          JSR FOO
9  |          LD  R7, SAVER7
10 | DONE     RET
11 | SAVER7   .FILL x0000
12 |          .END
```

- FOO infinitely loops for all R0 ≥ 3.
- Each call to FOO saves R7 in the same memory location.