

7.2.5 Register Usage

Waterloo C can use either of two register conventions: a very efficient implementation of the C run-time environment (the default), or an optional OS-style linkage convention. The OS-style convention is used when one of the OS, OSFUNC or OSENTRY compiler options is specified when the program is compiled.

The OS option generates code that assumes that register 12 (R12) is reserved for use as a run-time stack register and that the function base address is contained in R15 when the function is called. The OSENTRY option generates code that requires no run-time stack register and that makes no assumption about the function base address; each function defines a static save area. The OSFUNC option generates code that contains a call to a routine (\$GETSPTR) to retrieve a run-time stack pointer value; the function base address is assumed to be contained in R15.

When any of these options is specified, all functions defined and referenced in a source file are assumed to follow the particular linkage convention. An individual function can be identified as using a particular convention by adding the following preprocessor directive to the source file:

```
#pragma linkage <convention> <function>
```

<convention> can be either `clink`, `os`, `osfunc` or `osentry`. `clink` specifies that the default linkage convention is to be used for the specified function (even if an OS-style compiler option is specified when the source file in which the function is referenced is compiled); `os`, `osfunc` and `osentry` specify that the corresponding linkage is to be used for the indicated function.

For example,

```
#pragma linkage os name1
```

causes the compiler to generate OS-style linkage for the function `name1` which may be either defined in that source file or called from it.

```
#pragma linkage clink name2
```

causes the compiler to generate default-style linkage for the function `name2`.

The `#pragma linkage` directive allows various linkage conventions to be freely interspersed. Conversions from one convention to another are generated as needed.

The default-style linkage is the convention used in previous versions of Waterloo C and by the run-time library. Therefore, the use of OS-style linkage is only recommended for calls to existing assembler routines.

7.2.5.1 Default Linkage Convention

The following is a description of the register convention normally used by the Waterloo C compiler. This description is divided into three parts:

- the register usage at entry to a function and the function **prologue**,
- the register usage within a function, and
- the function exit process, called the function **epilogue**.

This convention is compatible with the linkage convention used in previous versions of Waterloo C and within the run-time library.

Function Entry

When a C function is called using the default linkage convention, it is assumed that the following registers contain the specified contents:

- R10 the run-time environment register (referred to as the **zeropage** register in previous versions, and abbreviated ZP), contains the address of register based global data and the end address of the run-time library read/write data area;
- R11 the linkage register (abbreviated LN), contains the address of the first executable instruction in the function;
- R12 the stack pointer (abbreviated SP), contains the address of parameters and unused run-time stack space;
- R13 the return address register (abbreviated RA), contains the address to which control will be transferred when the function returns.

A function's parameters are placed, in the order specified by the function call (but not necessarily the order evaluated), from low address to high address memory on the run-time stack allocated by the run-time library initialization routine `STARTUP`. At

Reference

entry to a function, the stack pointer (R12) contains the address of the function's first parameter. The address of the second parameter is obtained by adding the length of the first parameter to the contents of R12. Access to unused stack space is obtained by adding the total size of the function's parameters to the contents of R12.

Functions are called with a

```
BALR 13,11
```

instruction, which assigns R11 (LN) and R13 (RA) their initial values.

More information about the use of R10 is contained in the *Register-Based Global Data* and *Register-Based Global Data Using AUX Files* sections of this manual.

Function Prologue

To illustrate a function prologue, consider the HELLO program:

```
#include <stdio.h>

/* HELLO C -- A First Program. */

int main()
{
    printf( "hello, world\n" );
    return( 0 );
}
```

When it is compiled with the ASMSRC option, code similar to the following is written to the assembler file (DD ASM1):

```
@HELLO    CSECT
*        1: #include <stdio.h>
*        2:
*        3: /* HELLO C -- A First Program. */
*        4:
*        5: int main()
           ENTRY MAIN
MAIN      DS      0X
           STM     12,2,4(12)
           LR      15,12
           LA      12,32(,12)
           LR      14,11
```

```

        USING MAIN,14
*      6:  {
*      7:      printf( "hello, world\n" );
          LA      2,#G146
          ST      2,0(,12)
          L       11,#P89
          BALR    13,11
*      8:      return( 0 );
          SR      11,11
*      9:  }
          LM      12,2,4(15)
          BR      13
          DROP    14
          EXTRN   $CSTART
          ENTRY   $REF$CST
SREF$CST DS      0X
          DC      AL4($CSTART)
#G146    DS      0X
          DC      X'88859393966B40A6969993841500'
          EXTRN   PRINTF
#P89     DS      0F
          DC      AL4(PRINTF)
          END

```

The first instruction to be executed (STM) saves the registers that are used within the function at an unused location on the run-time stack. The next instruction copies the current value of the stack pointer into the activation record pointer, R15. R15 would be used within the function to access any parameters, auto variables, and saved register values. The activation record, pointed to by R15, is organized as follows:

AR->

parameters (low address)
...
auto variables
...
saved stack pointer (R12)
return address (R13)
saved base register (R14)
saved activation record (R15)
other saved registers (R0-R9)
...
(high address)

Reference

The next instruction in the prologue advances R12 to point to an unused run-time stack area. Parameters for called functions will be placed in this area.

The final prologue instruction in this example defines the function base register, R14. R11 is often used within the function as a temporary value and will be used during the function prologue to return any integer return value.

If a function consists of more than 4,095 bytes of code, an additional instruction is generated to load R1 with the address of the start of the function's data area.

In some cases, a function prologue may be slightly different. For example, if the compiler detects that no functions are called and no temporary data is required, R12 will be used directly instead of R15. Similarly, R11 may be used as the function base register instead of R14.

R1 may be used to access a function's read/write data if the `SPLIT` or `RENT` compiler option is specified. The sections *Separation Of Global Data* and *Register-Based Global Data* contain more information about these options.

Function Epilogue

When a function returns, instructions are executed to define a return value (if necessary), restore the saved register values and transfer control back to the caller, as specified by the return address. If the function return value is an integer or pointer type, the value is returned in R11. If the return value is a floating point, structure, or union type, the return value is returned in the address pointed to by R12 (that is, the parameter list is overwritten).

Summary

The following summarizes the register usage of the default linkage convention:

- F0–F6 floating point registers; saved by the function prologue and restored by the function epilogue if modified
- R0 intermediate result and temporary values
- R1 data area base register (as required)

- R2-R9 general purpose computations; saved by the function prologue and restored by the function epilogue if modified
- R10 (ZP) run-time environment data
- R11 (LN) linkage and return value
- R12 (SP) pointer to unused stack area
- R13 (RA) return address
- R14 (CR) code base register
- R15 (AR) activation record pointer

7.2.5.2 OS-Style Linkage Convention

The following is a description of the OS-style register convention, as generated by the OS, OSFUNC and OSENTRY compiler options and by the `#pragma linkage` directive.

Function Entry

When a function is called using an OS-style linkage convention, it is assumed that the following registers contain the specified contents. For the OSENTRY option, the following registers are reserved:

- R1 the parameter register (abbreviated PA), points to the list of parameters for the function;
- R10 the run-time environment register (referred to as the **zeropage** register in previous versions, and abbreviated ZP), contains the address of register based global data and the end address of the run-time library read/write data area;
- R13 the save area register (abbreviated SA), points to an 18-word register save area;
- R14 the return address register (abbreviated RA), contains the address to which control will be transferred when the function returns.

If the OSFUNC option is specified, the following registers are reserved:

- R1 (as above)
- R10 (as above)
- R13 (as above)
- R14 (as above)
- R15 the linkage register (abbreviated LN), contains the address of the first executable instruction in the function;

If the OS option is specified, the following registers are reserved:

R1 (as above)

R10 (as above)

R12 the stack pointer (abbreviated SP), contains the address of unused run-time stack space.

R13 (as above)

R14 (as above)

R15 (as above)

Functions that use an OS-style linkage convention are called with a

BALR 14,15

instruction, which assigns R14 (RA) and R15 (LN) their initial values.

More information about the use of R10 is contained in the *Register-Based Global Data* and *Register-Based Global Data Using AUX Files* sections of this manual.

Function Prologue

To illustrate a function prologue, consider the HELLO program:

```
#include <stdio.h>

/* HELLO C -- A First Program. */

int main()
{
    printf( "hello, world\n" );
    return( 0 );
}
```


Reference

When it is compiled with the OS and ASMSRC options, code similar to the following is written to the assembler file (DD ASM1):

```
@HELLO    CSECT
*        1: #include <stdio.h>
*        2:
*        3: /* HELLO C -- A First Program. */
*        4:
*        5: int main()
            ENTRY MAIN
MAIN      DS      0X
          STM     14,12,12(13)
          LR      9,12
          LR      11,15
          USING   MAIN,11
*        6: {
*        7:     printf( "hello, world\n" );
            LA     2,#G146
            ST     2,0(,12)
            L      15,#P89
            STM    12,1,4(12)
            STD    0,28(,12)
            LA     0,36(,12)
            ST     0,8(,13)
            LR     0,13
            LA     13,36(,12)
            ST     0,4(,13)
            SR     0,0
            ST     0,8(,13)
            LR     0,12
            LR     1,12
            LA     12,108(,12)
            BALR   14,15
            LR     12,0
            LM     0,1,20(12)
            LM     12,14,4(12)
            LD     0,28(,12)
*        8:     return( 0 );
            SR     15,15
*        9: }
            L      14,12(,13)
            LM     0,12,20(13)
            BR     14
            DROP   11
            EXTRN  $CSTART
            ENTRY  $REF$CST
$REF$CST DS      0X
          DC      AL4($CSTART)
```

```
#G146    DS    0X
          DC    X'88859393966B40A6969993841500'
          EXTRN PRINTF
#P89     DS    0F
          DC    AL4 (PRINTF)
          END
```

R9, the activation record pointer, is set to point to the unused stack area, initially pointed to by R12. All local variables are placed in this area and are accessed using R12.

R11 is used as the code base register (CR). Occasionally, R8 may be reserved as an alternate code base register as well.

R13 points to an 18-word register save area, whose space is allocated from the same stack as the activation record. The second word of this register save area is reserved for the address of the previous save area, the third word is reserved for the address of the next save area, and the fourth through 18th words are used to store the registers R14 through R12. (The second and third words are set up when another function is called; the first word is not used by Waterloo C.)

It is assumed that the free stack area is large enough to hold both the activation record and a register save area for any function called by the current function. (An overflow check can be included, if desired, to test this condition; see the section *Stack Overflow Checking* for more information.)

If HELLO is compiled with the OSFUNC and ASMSRC options, code similar to the following is written to the assembler file (DD ASM1):

```
@HELLO    CSECT
*          1: #include <stdio.h>
*          2:
*          3: /* HELLO C -- A First Program. */
*          4:
*          5: int main()
            ENTRY MAIN
MAIN       DS    0X
            USING MAIN,15
            STM   14,12,12(13)
            LR    2,11
            LR    3,13
            BALR  11,0
            B     8(,11)
            DC    AL4($GETSPTR)
```

Reference

```

L      11,4(,11)
BALR   13,11
L      12,0(,11)
LR     11,2
LR     13,3
LR     9,12
LR     11,15
DROP   15
USING  MAIN,11
*      6:  {
*      7:      printf( "hello, world\n" );
LA      2,#G146
ST      2,0(,12)
L      15,#P89
STM     12,1,4(12)
STD     0,28(,12)
LA      0,36(,12)
ST      0,8(,13)
LR      0,13
LA      13,36(,12)
ST      0,4(,13)
SR      0,0
ST      0,8(,13)
LR      0,12
LR      1,12
LA      12,108(,12)
EXTRN   $GETSPTR
LR      3,11
LR      4,13
BALR    11,0
B       8(,11)
DC      AL4($GETSPTR)
L      11,4(,11)
BALR    13,11
ST      12,0(,11)
LR      11,3
LR      13,4
BALR    14,15
LR      12,0
LM      0,1,20(12)
LM      12,14,4(12)
LD      0,28(,12)
*      8:      return( 0 );
*      SR      15,15
*      9:  }
LR      2,11
LR      3,13
BALR    11,0
B       8(,11)

```

```

        DC      AL4($GETSPTR)
        L        11,4(,11)
        BALR     13,11
        ST       12,0(,11)
        LR       11,2
        LR       13,3
        L        14,12(,13)
        LM       0,12,20(13)
        BR       14
        DROP     11
        EXTRN    $CSTART
        ENTRY    $REF$CST
$REF$CST DS      0X
        DC      AL4($CSTART)
#G146    DS      0X
        DC      X'88859393966B40A6969993841500'
        EXTRN    PRINTF
#P89     DS      0F
        DC      AL4(PRINTF)
        END

```

The code generated by the OSFUNC option is similar to the code generated by the OS option and the only significant difference is that for OSFUNC, SP, the stack pointer register, is loaded by calling the \$GETSPTR routine. With the OS option, it is assumed that R12 already points to an unused stack area.

If HELLO is compiled with the OSENTRY and ASMSRC options, code similar to the following is written to the assembler file (DD ASM1):

```

@HELLO  CSECT
*       1: #include <stdio.h>
*       2:
*       3: /* HELLO C -- A First Program. */
*       4:
*       5: int main()
        ENTRY MAIN
MAIN     DS      0X
        USING   MAIN,15
        STM     14,12,12(13)
        BALR    15,0
        B       8(,15)
        DC      X'00000006'
        S       15,4(,15)
        BALR    12,0
        B       8(,12)
        DC      AL4(#G150)
        L       12,4(,12)

```

Reference

```

        LR      9,12
        LR      11,15
        DROP    15
        USING   MAIN,11
*       6:      {
*       7:      printf( "hello, world\n" );
        LA      2,#G146
        ST      2,0(,12)
        L       15,#P89
        STM     12,1,4(12)
        STD     0,28(,12)
        LA      0,36(,12)
        ST      0,8(,13)
        LR      0,13
        LA      13,36(,12)
        ST      0,4(,13)
        SR      0,0
        ST      0,8(,13)
        LR      0,12
        LR      1,12
        LA      12,108(,12)
        BALR    14,15
        LR      12,0
        LM      0,1,20(12)
        LM      12,14,4(12)
        LD      0,28(,12)
*       8:      return( 0 );
        SR      15,15
*       9:      }
        L       14,12(,13)
        LM      0,12,20(13)
        BR      14
        DROP    11
        EXTRN   $CSTART
        ENTRY   $REF$CST
$REF$CST DS    0X
        DC      AL4($CSTART)
#G146     DS    0X
        DC      X'88859393966B40A6969993841500'
        EXTRN   PRINTF
#P89     DS    0F
        DC      AL4(PRINTF)
#G150     DS    0X
        DS      196X
        END

```

The main difference between the `OSENTRY` option and the other OS options is that with `OSENTRY`, R15 is loaded with the code start address and R12 is set to point to a

fixed (static) block of memory. Recursive functions cannot be compiled with this option.

The preceding examples are provided only for example purposes. Because they define and reference symbols (`main` and `printf`) that are referred to by the run-time library, they cannot be linked and run unless suitable library interface routines are provided.

Function Epilogue

When an OS-style function returns, the original values of R14 and R0 through R12 are restored from the register save area pointed to by R13. The function's return value (or return code) is loaded into R15 if it is of integer or pointer type. If it is of floating point type, it is loaded into floating point register 0. If it is a structure or union, it is stored at the address pointed to by R12. A branch to the address stored in R14 is then executed.

Summary

The following summarizes the register usage of the OS-style linkage convention:

- F0 floating point return value
- F2–F6 floating point registers; saved by the function prologue and restored by the function epilogue if modified
- R0 intermediate result and temporary values
- R1 (PA) parameter and auto variable area
- R2–R7 general purpose computations; saved by the function prologue and restored by the function epilogue if modified
- R8 data register, if necessary
- R9 (AR) activation record pointer
- R10 (ZP) run-time environment data

Reference

R11	(CR) code base register
R12	(SP) pointer to unused stack area
R13	(SA) 18-word save area
R14	(RA) return address
R15	(LN) linkage register and return code