Computer Structure and Language

Hamid Sarbazi-Azad
Department of Computer Engineering
Sharif University of Technology (SUT)
Tehran, Iran

Base Register Definition and Initialization

We use directive: **USING BaseAddress, BR**

to tell to assembler that base register is BR and its content is BaseAddress. Note that this directive only informs assembler and does not initialize the base register BR. → It is the duty of programmer to do so.

For example:

HERE USING HERE,12

Tells the assembler that base address is HERE and base register is R12. So, assembler generates the address for symbol LABEL as:

$$(R12) + \underbrace{LABEL - HERE}_{\text{Displacement}}$$

To initialize the base register, we usually use a specific instruction that is also used for subroutine call (BALR instruction).

RR Instructions:

OPCODE	r1	r2

Branch and Link Register

Mnemonic: BALR r1,r2
Operation: r1 ← (PC); PC ← (r2);
OPCODE: 05h

Example 1:

LA 2,ADD ADD is a label
Assembly instruction: BALR 3,2
Operation: R3 ← (PC); PC ← (R2);
Machine code: 0532

Example 2:
Assembly instruction: BALR 12,0
Operation: R12 ← (PC);
Machine code: 05C0

Base Register Definition and Initialization

Now, we can define base register and initialize it properly as:

```
PROG  START 0
      BALR  12,0
HERE  USING HERE,12
      ….
```

Above code defines R12 as the base register and initializes it to 2.

Is it OK if we write?

```
PROG  START 0
HERE  USING HERE,12
      BALR  12,0
      ….
```

NO, we have told assembler that base register content is 0 but have initialized it to 2. → All displacements are calculated by a skew of 2.

Base Register Definition and Initialization

What about below code ?

```
PROG  START  0
HERE  USING  HERE+2,12
      BALR   12,0
      ….
```

YES, it is OK. Base register is initialize to 2 and assembler knows that the base address is 2. ☺

Why not below code?

```
PROG  START  0
HERE  USING  HERE,12
      LA     12,HERE
```

NO, we do not have a base register is initialized to generate address HERE in LA instruction?

Base Register Definition and Initialization

Note that if we use * as a symbol in assembler instructions, it mean the Location Counter (address of current instruction).

So, the popular way to define base register is usually:

```
PROG  START  0
      BALR   12,0
      USING  *,12
      ….
```

Above code defines R12 as the base register and initializes it to 2.

Is the following code OK?

```
PROG  START  0
      USING  *+2,12
      BALR   12,0
      ….
```

Yes, it is OK.

Base Register Definition and Initialization

What if we need more than one segment?
We can use directive USING as: USING BaseAddress,r1,r2,…

For example, below code defines 3 base registers R12, R11 and R9 in order.

```
        BALR    12,0
        USING   *,12,11,9
        LA      11,4095(12)
        LA      11,1(11)
        LA      9,4095(11)
        LA      9,1(9)
```

The address of symbols in range:
BaseAddress…BaseAddress+4095 are generated by base register R12,
BaseAddress+4096…BaseAddress+8191 are generated by base register R11,
BaseAddress+8192…BaseAddress+12287 are generated by base register R9.

Base Register Definition and Initialization

Alternative way to define 3 base registers R12, R11, and R9 in order, is:

```
        BALR    12,0
        USING   HERE,12,11,9
HERE    L       11,SEGMENT2
        L       9,SEGMENT3
        ….
```

If we have the following variable definitions in the first segment:

```
SEGMENT2   DC     A(HERE+4096)
SEGMENT3   DC     A(HERE+8192)
```

Note: Type **A** defines a full-word that contains an address.

Base Register Definition and Initialization

The following code defines 4 base registers R9, R10, R11, R12 in order:

```
         BALR    9,0
         USING   HERE,9,10,11,12
HERE     LM      10,12,SEGMENTS

         ….

SEGMENTS     DC      A(HERE+4096,HERE+8192,HERE+12288)
         ….
```


Subroutines

Subroutine calls can be done by BALR instruction where r1 keeps a copy of PC (to be used at the end of subroutine for return) and fills PC with the address of the first instruction of the subroutine which is in r2.

Another instruction in RX format can be used to directly call a subroutine (no need to copy its address into register r2 and then use BALR r1,r2).

It is BAL instruction.

Subroutine Call (RX Format):

OPCODE	r1	X2	B2		D2

Branch and Link

Mnemonic:	BAL	r1,S2(X2)
	BAL	r1,S2
	BAL	r1,D2(X2,B2)
	BAL	r1,D2(X2)
	BAL	r1,D2(,B2)

Operation: r1 ← (PC); PC ← (B2)+(X2)+D2;

OPCODE: 45h

Subroutine Call (RX Format):

OPCODE	r1	X2	B2		D2

Branch and Link

Example 1:
Assembly instruction: BAL 3,SORT SORT is a label = (R12)+100
Operation: R3 ← (PC); PC ← (R12)+100;
Machine code: 4530C064

Example 2:
Assembly instruction: BAL 2,0(3,4)
Operation: R2 ← (PC); PC ← (R3)+(R4);
Machine code: 45234000

Example 3:
Assembly instruction: BAL 3,0(2) equivalent to BALR 3,2
Operation: R3 ← (PC); PC ← (R2);
Machine code: 45320000

Subroutines

In IBM system, the following convention is agreed and used by all programmers for writing programs/subroutines calling subroutines.

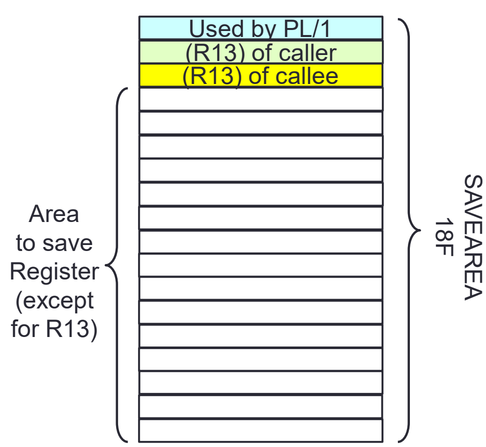
R0: used to pass a single input argument to called subroutine.
R1: used to pass a single output argument to caller, or to pass the address of the arguments list of addresses.
R13: used to keep the address of the save area provided by caller to save register contents by the called subroutine.
R14: used to keep the return address to caller.
R15: used to keep the called subroutine address.

So, registers R2 to R12 are available to use.

Subroutines

The save area contains 18 full-words arranged as:

This area must be defined by caller but used to save registers contents by callee.



Subroutines

Duty of caller and callee.

Subroutine 1

- Save the registers into the SaveArea provided by its caller
- Defines SaveArea and puts its address in R13

To call a subroutine:

- Puts the input argument into R0 (or pointer to arguments' address list into R1)
- Put the address of subroutine in R15
- Calls subroutine (keep return address in R14)

At the end:

- Restore registers
- Returns to address in R14

R13

Save Area (18F)

Subroutine 2

- Save the registers into the SaveArea provided by its caller
- Defines SaveArea and puts its address in R13

To call a subroutine:

- Puts the input argument into R0 (or pointer to arguments' address list into R1)
- Put the address of subroutine in R15
- Calls subroutine (keep return address in R14)

At the end:

- Restore registers
- Returns to address in R14

R13

Save Area (18F)

Subroutine 3

- Save the registers into the SaveArea provided by its caller

At the end:

- Restore registers
- Returns to address in R14

This is a terminal subroutine that does not call any other subroutine.
→ No need to have SaveArea

Example 1: Write an assembly program to call a subroutine to calculate factorial of a given number (local subroutine; one assembly file).

```
MAIN    START   0
        STM     14,12,12(13)
        BALR    12,0
        USING   *,12
        ST      13,SAVEAREA+4
        LA      13,SAVEAREA
        L       0,N
        LA      15,FACT   }  BAL  14,FACT
        BALR    14,15     }
        ST      1,FACT_N
        L       13,SAVEAREA+4
        LM      14,12,12(13)
        BR      14
```

```
FACT    STM     14,12,12(13)
        LA      3,1
        LR      4,0
LOOP    C       4,=H'0'
        BNH     OUT
        MR      2,4
        BCTR    4,0
        B       LOOP
OUT     LM      14,2,12(13)
        LR      1,3
        LM      3,12,32(13)
        BR      14

SAVEAREA        DS      18F
N               DC      F'5'
FACT_N          DS      F
        END     MAIN
```

Example 2: Write an assembly program to call a subroutine to calculate factorial of a given number (external subroutine; two assembly files).

```
MAIN      START  0
          STM    14,12,12(13)
          BALR   12,0
          USING  *,12
          ST     13,SAVEAREA+4
          LA     13,SAVEAREA
          L      0,N
          L      15,FACTADDR
          BALR   14,15
          ST     1,FACT_N
          L      13,SAVEAREA+4
          LM     14,12,12(13)
          BR     14
SAVEAREA         DS     18F
N                DC     F'5'
FACT_N           DS     F
FACTADDR         DC     V(FACT)
          END    MAIN
```

```
FACT      START  0
          STM    14,12,12(13)
          BALR   12,0
          USING  *,12
          LA     3,1
          LR     4,0
LOOP      C      4,=H'0'
          BNH    OUT
          MR     2,4
          BCTR   4,0
          B      LOOP
OUT       LM     14,2,12(13)
          LR     1,3
          LM     3,12,32(13)
          BR     14
          END    FACT
```


Example 3:

Write an assembly program to call a subroutine to calculate Σ(ARRAY[i]!) for array ARRAY of N positive halfwords. Suppose we have external subroutine FACT in Example 2.

```
MAIN    START  0
        STM    14,12,12(13)
        BALR   12,0
        USING  *,12
        ST     13,SAVEA+4
        LA     13,SAVEA
        LA     1,ARGS
        L      15,=V(ADDER)
        BALR   14,15
        L      13,SAVEA+4
        LM     14,12,12(13)
        BR     14
SAVEA   DS     18F
N       DC     H'5'
ARRAY   DC     H'3,2,7,4,5'
SUM     DS     F
ARGS    DC     V(N,ARRAY,SUM)
        END    MAIN
```

```
ADDER   START  0
        STM    14,12,12(13)
        BALR   12,0
        USING  *,12
        ST     13,SAVEA+4
        LA     13,SAVEA
        L      2,0(1)     @N
        LH     2,0(2)     N
        L      3,4(1)     @ARRAY
        L      5,8(1)     @SUM
        XR     4,4
LOOP    LH     0,0(3)
        BAL    14,=V(FACT)
        AR     4,1
        LA     3,2(3)
        BCT    2,LOOP
        ST     4,0(5)     Store in SUM
        L      13,SAVEA+4
        LM     14,12,12(13)
        BR     14
SAVEA   DS     18F
        END    ADDER
```

Example 3:

Write an assembly program to call a subroutine to calculate Σ(ARRAY[i]!) for array ARRAY of N positive halfwords. Suppose we have external subroutine FACT in Example 2.

```
MAIN    START   0
        STM     14,12,12(13)
        BALR    12,0
        USING   *,12
        ST      13,SAVEA+4
        LA      13,SAVEA
        LA      1,ARGS
        L       15,=V(ADDER)
        BALR    14,15
        L       13,SAVEA+4
        LM
        BR
SAVEA   DS
N       DC
ARRAY   DC
SUM     DS      F
ARGS    DC      V(N,ARRAY,SUM)
        END     MAIN
```

```
ADDER   START   0
        STM     14,12,12(13)
        BALR    12,0
        USING   *,12
        ST      13,SAVEA+4
        LA      13,SAVEA
        L       2,0(1)        @N
        LH      2,0(2)        N
        L       3,4(1)        @ARRAY
        L       5,8(1)        @SUM
        XR      4,4
LOOP    LH      0,0(3)
                14,=V(FACT)
                4,1
                3,2(3)
                2,LOOP
                4,0(5)      Store in SUM
                13,SAVEA+4
                14,12,12(13)
        BR      14
SAVEA   DS      18F
        END     ADDER
```

Homework:
Generate the machine code of this program.

End of Slides