Lenguajes de Interfaz

Unidad 1
Introducción al Lenguaje Ensamblador

M. C. Miguelangel Fraga Aguilar

1.1 Importancia de la programación en Lenguaje Ensamblador

- Mejor conocimiento del funcionamiento del CPU
- Implementar operaciones que son difíciles de programar en alto nivel
  - Aritmética extendida, operaciones no disponibles en c (bit reversal), uso de instrucciones especiales
- Optimizar el desempeño de código que se ejecuta muchas veces en un programa

1.2 El procesador y sus registros internos

- Los registros no son localidades de memoria, están construidos con FilFlops y son parte del camino de datos del CPU
- Casi todas las instrucciones requieren del uso de un registro y algunas solo funcionan con un registro en especifico
- La lectura y escritura de los registros es lo más rápido en un procesador

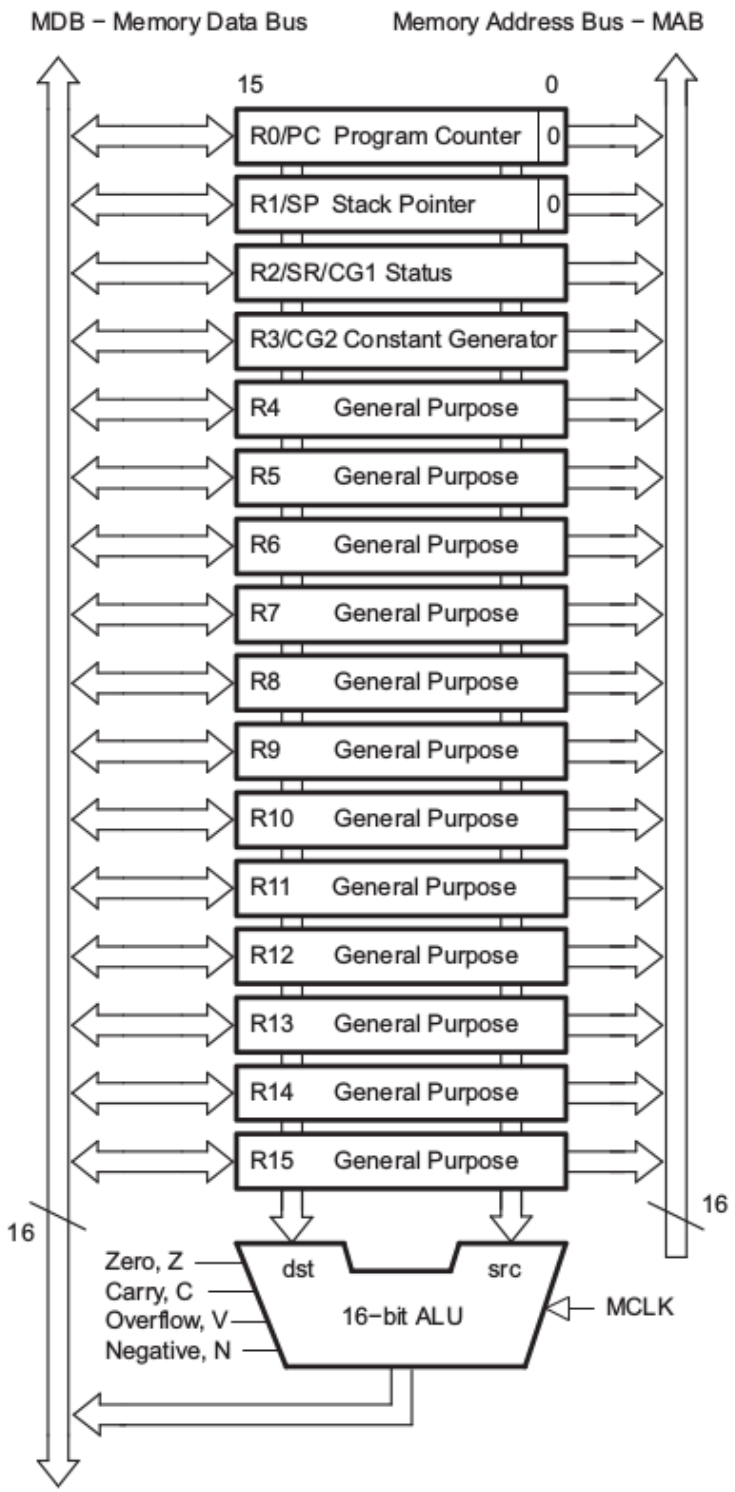


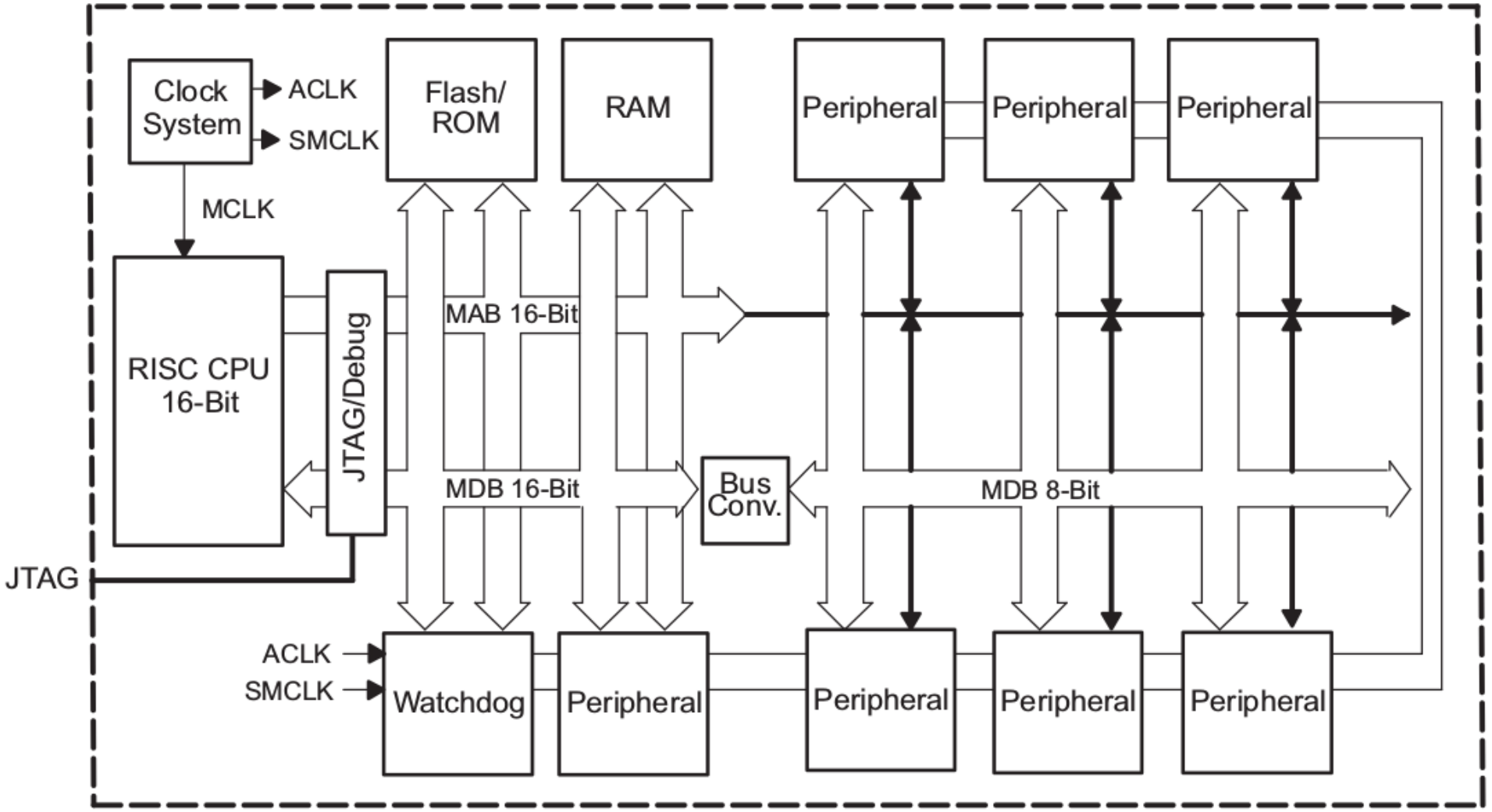
Diagrama a bloques del CPU

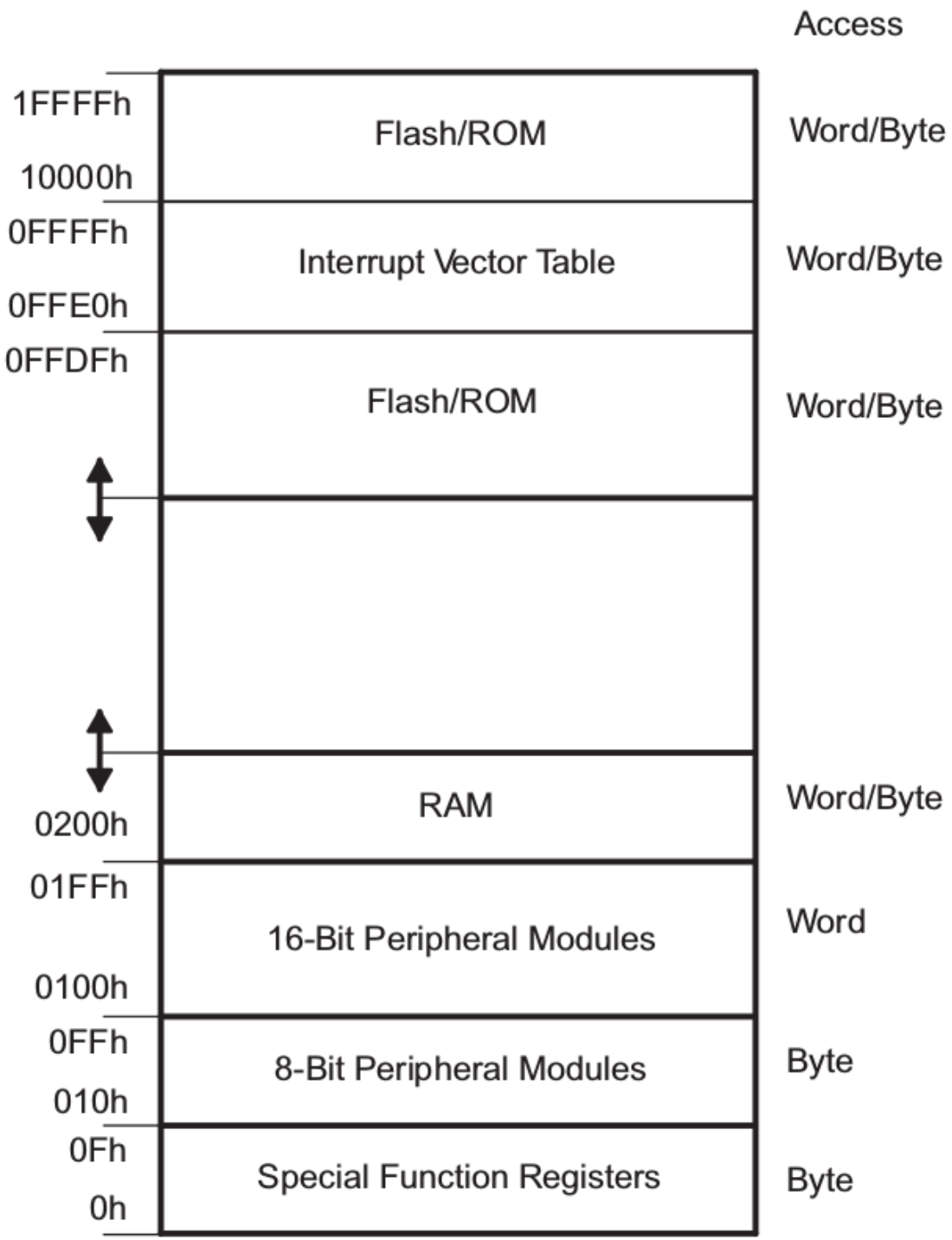
De: Texas Instruments. MSP430x2xx Family User's Guide. SLAU144J 2013

Registros

- PC/R0 Guarda la dirección de la siguiente instrucción a buscar
- SP/R1 Apuntador a la Pila. La pila solo almacena palabras completas y trabaja con predecremento / postincremento.
- SR/R2 Registro de estado. Contiene las banderas de estado y de control
- Generadores de constantes

1.2 Memoria


Clock System
ACLK
SMCLK
MCLK
Flash/ ROM
RAM
Peripheral
Peripheral
Peripheral
RISC CPU 16-Bit
JTAG/Debug
MAB 16-Bit
MDB 16-Bit
Bus Conv.
MDB 8-Bit
JTAG
ACLK
SMCLK
Watchdog
Peripheral
Peripheral
Peripheral
Peripheral



Mapa de memoria

Tipos de memoria

- Flash/ROM: Puede usarse para almacenar tanto código como datos. Se puede acceder tanto por bytes como por palabras. Usualmente se trabaja como memoria de solo lectura. No es volátil
- RAM: Aunque puede usarse tanto para código como para datos, es raro usarla para código. Es volátil y al arrancar el programa tiene valor aleatorios, por lo que debe asignarse un valor inicial a las variables en RAM.

Periféricos

- Los periféricos se encuentran mapeados en memoria, por lo que no se requiere instrucciones especiales para acceder a ellos.
- La mayoría de los registros de los periféricos se encuentra en el rango de 0100h a 01FFh y solo se pueden acceder con instrucciones de 16 bits
- Registros de función especial - de 0000h a 000Fh. Solo se pueden acceder con instrucciones de 8 bits. Ver hoja de datos MSP430G2553

Interrupciones

- Es una llamada a subrutina hecha por el hardware.
- La rutina llamada se conoce como rutina de atención de interrupción
- Un periférico puede tener una o mas interrupciones asociadas
- La dirección de inicio de la rutina se almacena en una dirección de memoria especifica, llamada vector de interrupción

Tabla de vectores de interrupción

- Todos los vectores de interrupción se encuentran agrupados en la tabla de vectores de interrupción
- Cada vector es de 16 bits
- Tambien se encuentra el vector de reset en la tabla de vectores de interrupciones. Este contiene la dirección de la primera instrucción a ejecutar

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-Up External Reset Watchdog Timer+ Flash key violation PC out-of-range(1)	PORIFG RSTIFG WDTIFG KEYV(2)	Reset	0FFFEh	31, highest
NMI Oscillator fault Flash memory access violation	NMIIFG OFIFG ACCVIFG(2)(3)	(non)-maskable (non)-maskable (non)-maskable	0FFFCh	30
Timer1_A3	TA1CCR0 CCIFG(4)	maskable	0FFFAh	29
Timer1_A3	TA1CCR2 TA1CCR1 CCIFG, TAIFG(2)(4)	maskable	0FFF8h	28
Comparator_A+	CAIFG(4)	maskable	0FFF6h	27
Watchdog Timer+	WDTIFG	maskable	0FFF4h	26
Timer0_A3	TA0CCR0 CCIFG(4)	maskable	0FFF2h	25
Timer0_A3	TA0CCR2 TA0CCR1 CCIFG, TAIFG (5)(4)	maskable	0FFF0h	24
USCI_A0/USCI_B0 receive USCI_B0 I2C status	UCA0RXIFG, UCB0RXIFG(2)(5)	maskable	0FFEEh	23
USCI_A0/USCI_B0 transmit USCI_B0 I2C receive/transmit	UCA0TXIFG, UCB0TXIFG(2)(6)	maskable	0FFECh	22
ADC10 (MSP430G2x53 only)	ADC10IFG(4)	maskable	0FFEAh	21
			0FFE8h	20
I/O Port P2 (up to eight flags)	P2IFG.0 to P2IFG.7(2)(4)	maskable	0FFE6h	19
I/O Port P1 (up to eight flags)	P1IFG.0 to P1IFG.7(2)(4)	maskable	0FFE4h	18
			0FFE2h	17
			0FFE0h	16
See (7)			0FFDEh	15
See (8)			0FFDEh to 0FFC0h	14 to 0, lowest

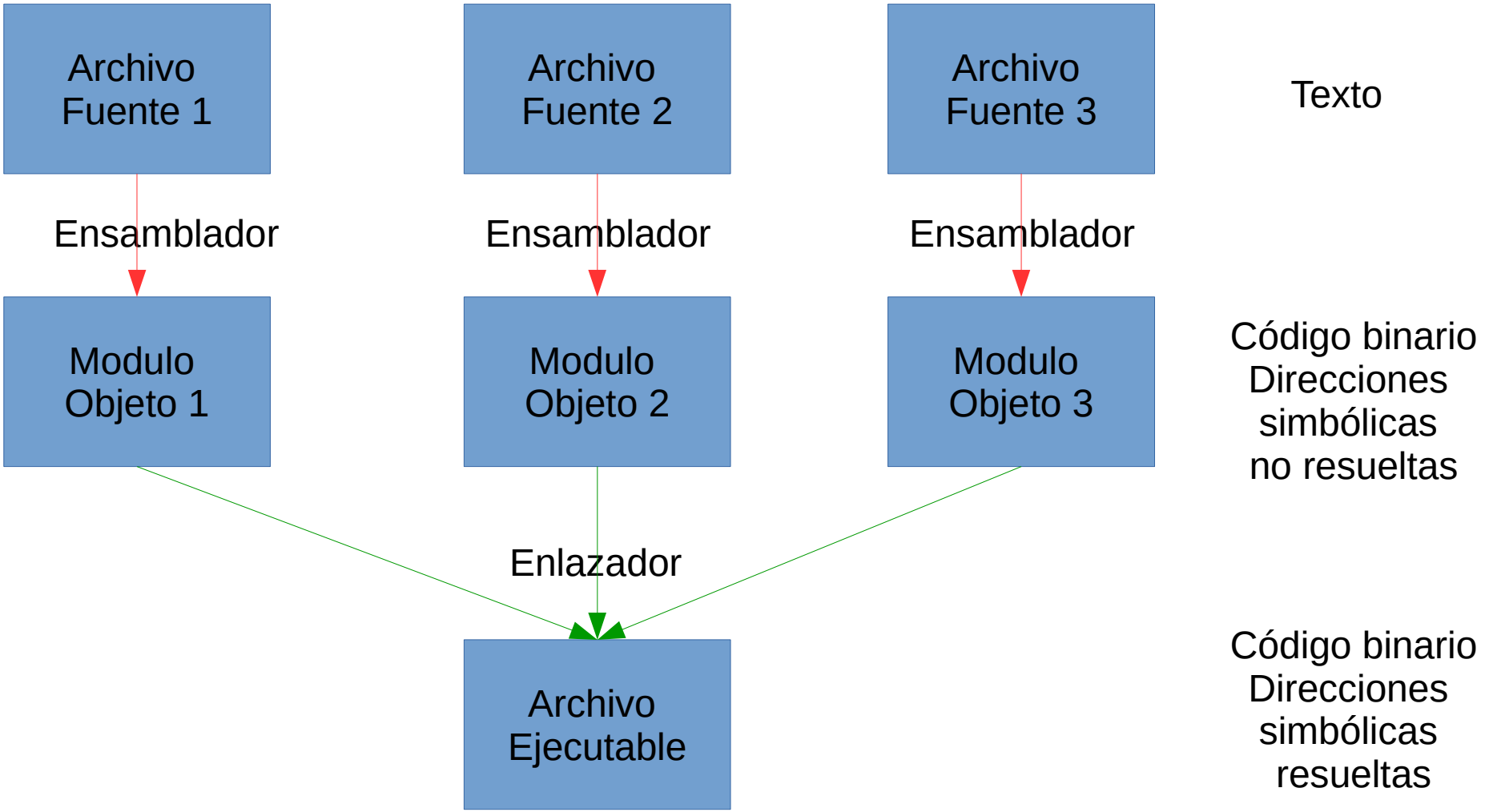
Esqueleto de un programa en ensamblador para MSP430

```
#include "msp430.h"                     ; #define controlled include file

        NAME    main                    ; module name

        PUBLIC  main                    ; make the main label vissible
                                        ; outside this module
        ORG     0FFFEh
        DC16    init                    ; set reset vector to 'init' label

        RSEG    CSTACK                  ; pre-declaration of segment
        RSEG    CODE                    ; place program in 'CODE' segment

init:   MOV     #SFE(CSTACK), SP        ; set up stack

main:   NOP                             ; main program
        MOV.W   #WDTPW+WDTHOLD,&WDTCTL  ; Stop watchdog timer
        JMP $                           ; jump to current location '$'
                                        ; (endless loop)
        END
```

Directivas del ensamblador

- No son instrucciones del procesador, son ordenes para el ensamblador
- ORG – indica la dirección de memoria en la que se va a almacenar el resultado de las siguientes lineas
- DC16 – almacena una constante de 16 bits
- Etiquetas. Cualquier palabra no reservada que comiense en la columna cero y que termine en : se llama una etiqueta. Es un nombre simbolico para la dirección de memoria en donde se almacena el resultado de la instrucción o directiva de dicha linea.
- Es conveniente usar etiquetas para referirse a las direcciones de memoria de variables o instrucciones

Proceso de ensamblado y ligado (Enlazado)



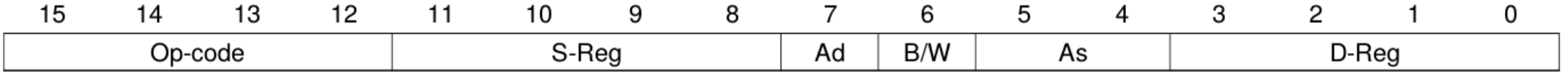
Código relocalizable: puede ejecutarse en diferentes direcciones de memoria

Ensamblado y enlazado

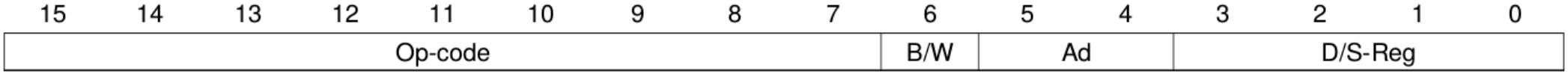
- Compilación (C C++) o ensamblado (asm) producen un archivo de código objeto.
- El código objeto ya incluye las instrucciones en binario, pero no todas las etiquetas han sido resueltas a una dirección fija.
- El enlazador une los archivos de código objeto y las librerías que se necesitan para formar el código ejecutable. Se resuelven todas las direcciones.
- Libreria: solo se incluyen las funciones que se usan.

Formatos de instrucciones

- Formato I – instrucciones con dos operandos



- Formato II – Instrucciones con un operando



- Saltos condicionales

Modos de direccionamiento

As/Ad	Modo	Sintaxis	Descripción
00/0	Registro	Rn	El operando es el contenido del registro
01/1	Indexado	X(Rn)	(Rn+X) apuntan al operando. X se almacena en la siguiente palabra
01/1	Simbólico	ADDR	(PC+X) apuntan al operando. X se almacena en la siguiente palabra
01/1	Absoluto	&ADDR	La palabra siguiente a la instrucción contiene la dirección absoluta. X se almacena en la siguiente palabra. Se usa el modo indexado X(SR)
10/-	Indirecto por registro	@Rn	Rn es usado como apuntador al operando
11/-	Indirecto con auto incremento	@Rn+	Rn es usado como apuntador al operando. Rn es incrementado en 1 para instrucciones .b y en 2 para instrucciones. w
11/-	Inmediato	#N	La palabra siguiente a la instrucción contiene la constante inmediata N. Se usa el modo indirecto con auto incremento @PC+

Ejemplos en guía del usuario

Otras directivas

- RSEG – Marca el inicio de una segmento relocalizable
  - CSTACK – segmento de pila rw - 0x3B0 – 0x3FF
  - CODE - Segmento de codigo r – 0xC000- 0xFF00
  - DATA16_N -segmento de datos rw – 0x200-0x3AF
- DS8 DS16 DS32 DS64 apartan espacio no inicializado – Equivalente a declarar la variable sin valor inicial

Declaración de Variables

Variables sin valor inicial

```
        RSEG    DATA16_N                ;segmento de datos no inicializados
                                        ;Se resuelve a RAM
var16:  DS16    4
var8:   DS8     1

        RSEG    CSTACK                  ; pre-declaration of segment

        RSEG    CODE                    ; place program in 'CODE' segment
                                        ; Se resulelve a FLASH
con16:  DC16    1,2,3,4
con8:   DC8     5,6,7,8

init:   MOV     #SFE(CSTACK), SP        ; set up stack

main:   NOP                             ; main program
        MOV.W   #WDTPW+WDTHOLD,&WDTCTL  ; Stop watchdog timer
        MOV     con16,R4
```

Constantes con valor inicial

Operaciones aritméticas

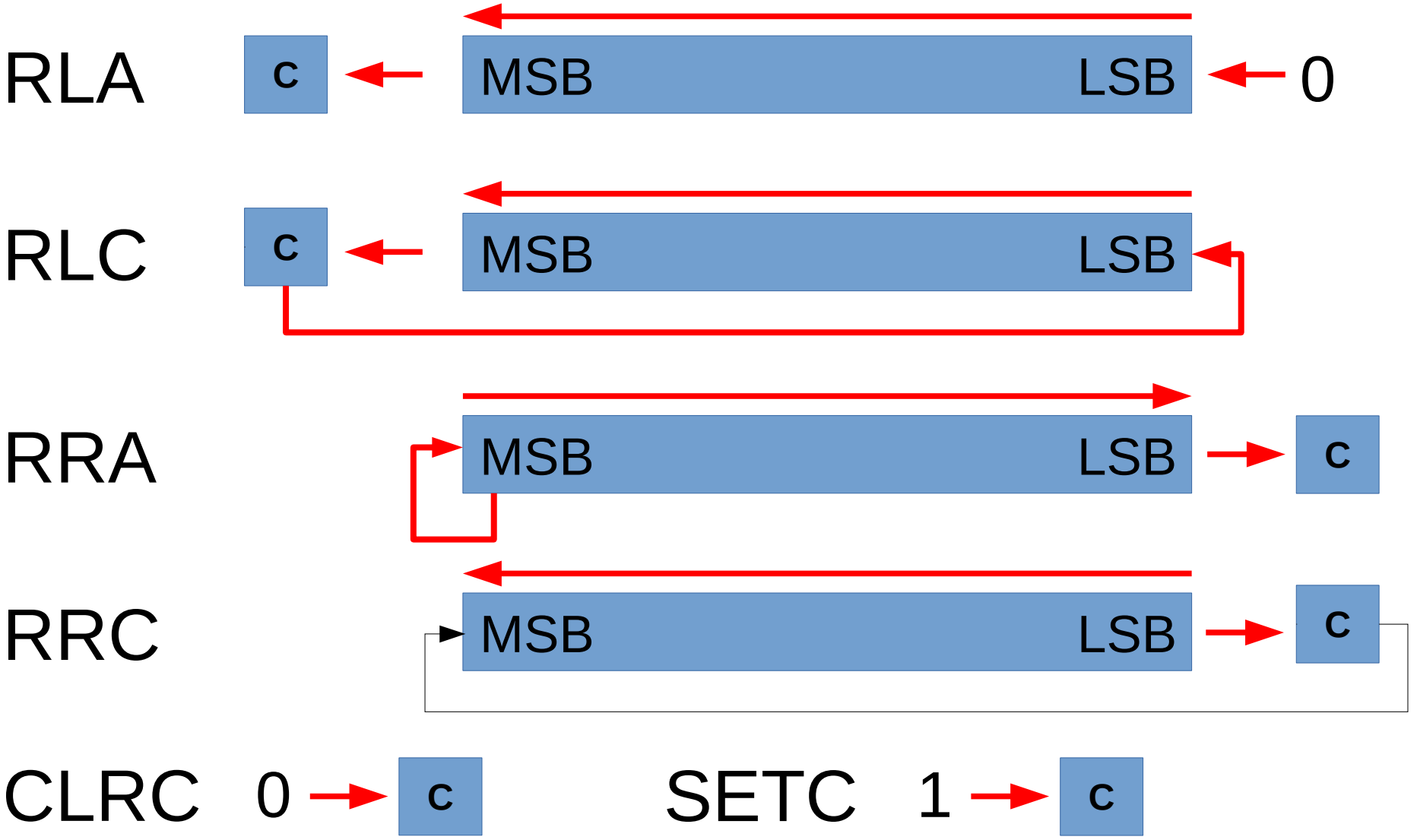
- Recordar la prioridad lógica aritmética
- MOV fuente, destino fuente → destino
  No modifica banderas
- ADD S,D S+D → D Modifica C V N Z
- ADDC S,D S+D+C → D Modifica C V N Z
- SUB S,D D-S → D Modifica C V N Z
- SUBC S,D D-S-C → D Modifica C V N Z
- INC D D+1 → D Modifica C V N Z
- DEC D D-1 → D Modifica C V N Z

Nota: Como la resta se hace sacando el complemento a dos del sustraendo y sumando, la bandera de acarreo es igual al negado del préstamo de la resta

Suma multipalabra

```
        RSEG    CSTACK                  ; pre-declaration of segment
        RSEG    CODE                    ; place program in 'CODE' segment
cop1:    DC32    0xfedcba98              ;operando 1
cop2:    DC32    0x76543210              ;operando 2

init:   MOV     #SFE(CSTACK), SP        ; set up stack

main:   NOP                             ; main program
        MOV.W   #WDTPW+WDTHOLD,&WDTCTL  ; Stop watchdog timer

        MOV     cop1,R4                 ;Suma la palabra baja
        ADD     cop2,R4
        MOV     R4,res                  ;Almacena el resultado
        MOV     cop1+2,R5                 ;Suma la palabra baja
        ADDC    cop2+2,R5
        MOV     R5,res+2                  ;Almacena el resultado

        JMP $                           ; jump to current location '$'
                                        ; (endless loop)
```

Rotaciones


RLA
c
MSB
LSB
0
RLC
c
MSB
LSB
RRA
MSB
LSB
c
RRC
MSB
LSB
c
CLRC
0
c
SETC
1
c

Multiplicación por corrimientos y sumas

- Recordar que una multiplicación es una suma abreviada. 5*a=a+a+a+a+a
- Un corrimiento a la izquierda equivale a una multiplicación por 2
- Con corrimientos a la izquierda se puede multiplicar por cualquier entero positivo. Ejemplo: 7*a=(a<<1+a)<<1+a=a<<2+a<<1+a
- Multiplicar por un entero negativo implica multiplicar por su valor absoluto y sacar el complemento a dos

Multiplicador por Hardware

- Periférico externo al CPU capaz de realizar multiplicación sin signo, con signo, Multiplica y acumula con signo y sin signo
- Puede trabajar con todas las combinaciones de operandos de 8 y 16 bits
- La operación que realiza depende de donde se almacene el primer operando
- El resultado esta listo tres ciclos de reloj después de escribir el operando 2

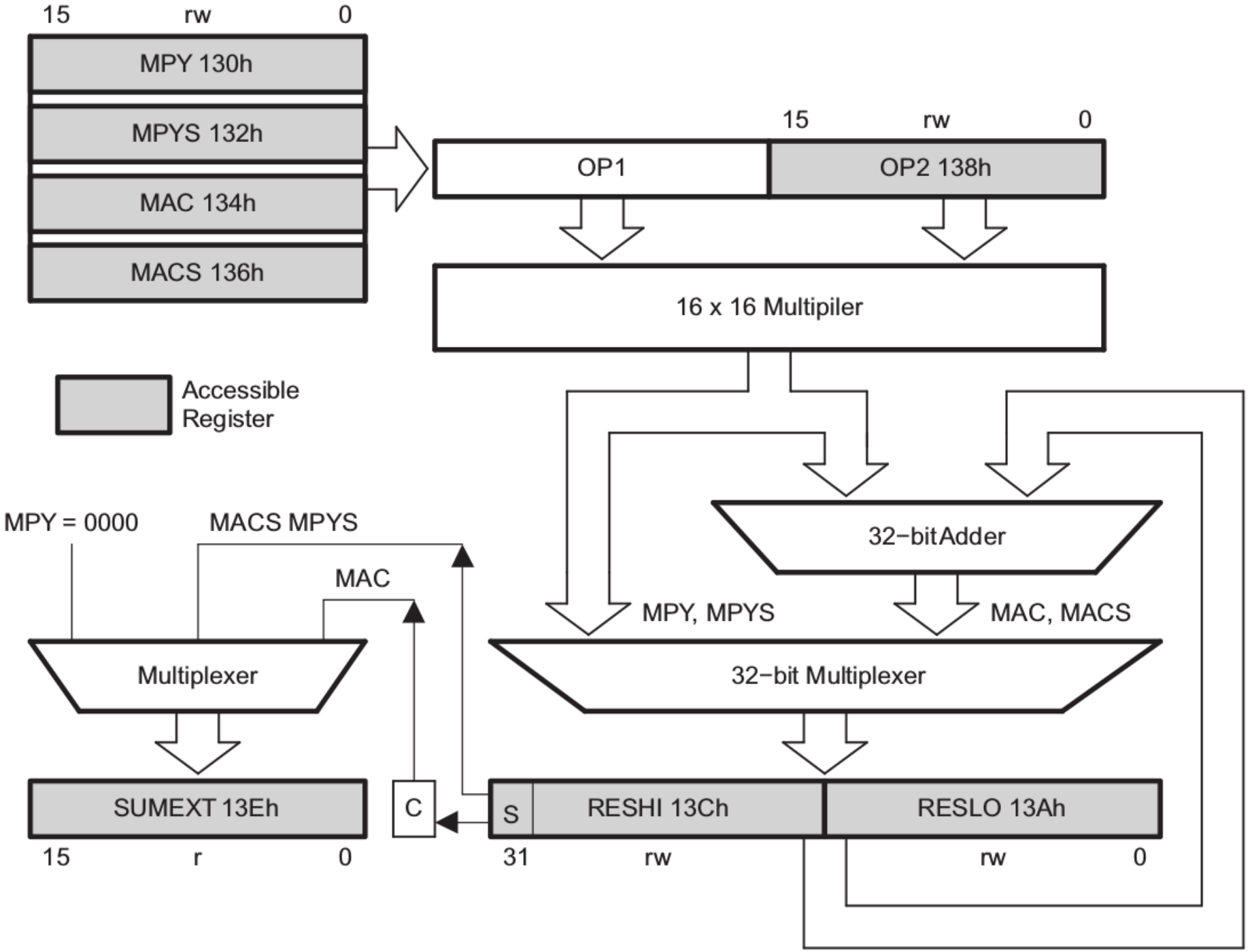


Figure 11-1. Hardware Multiplier Block Diagram