Example: Decimal to hex (binary) conversion using the addition

Use a 4-digit decimal number

Objectives of Example

- Illustrate how few instructions are all we need for apparent complicated situations
- Illustrate initial process
- Introduce the concept of MACRO (Section 4.6, page 198)
  - A macro is a set of instructions grouped under a user-defined name
- Introduce an example of using stack and addressing mode @SP+ if no pop is used.

Previous considerations

- Decimal means BCD
  - If data is 247h, conversion yields 00F7h
- What do we have to work?
  - A microcontroller.
  - Theory…!!!!
- Basic principle:
- $(N_3N_2N_1N_0)_{10}$=N3*A^3+N2*A^2+N1*A+N0

To simplify calculations: Modify formula!

- N3*A^3+N2*A^2+N1*A+N0=
  = (N3*A+N2)*A^2+N1*A+N0 =
  = ((N3*A+N2)*A + N1)*A + N0

****

P=N3*A → Extract N3, multiply by A

P=P+N2 → Extract N2, add to P

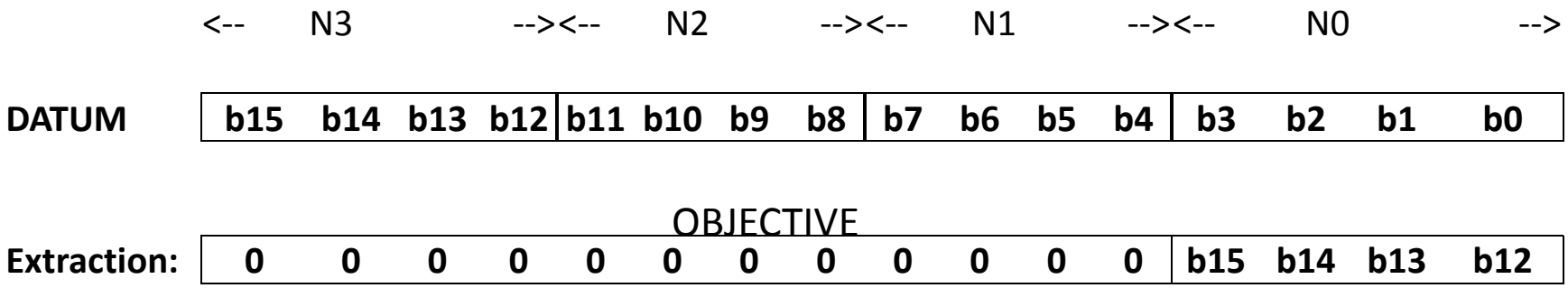
P=P*A → Multiply P by 10

P=P+N1 → Extract N1, add to P

P=P*A → Multiply P by A

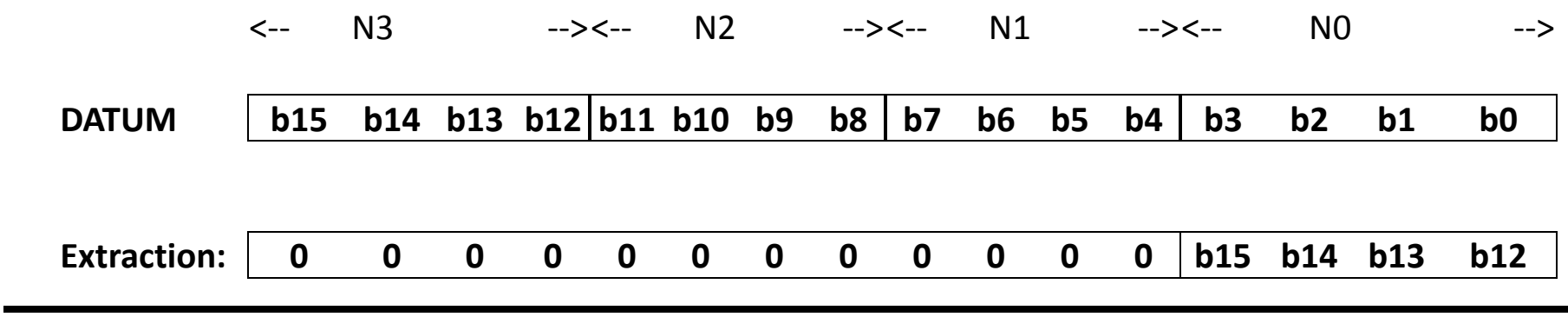
P=P+N0 → Extract N0, add to P

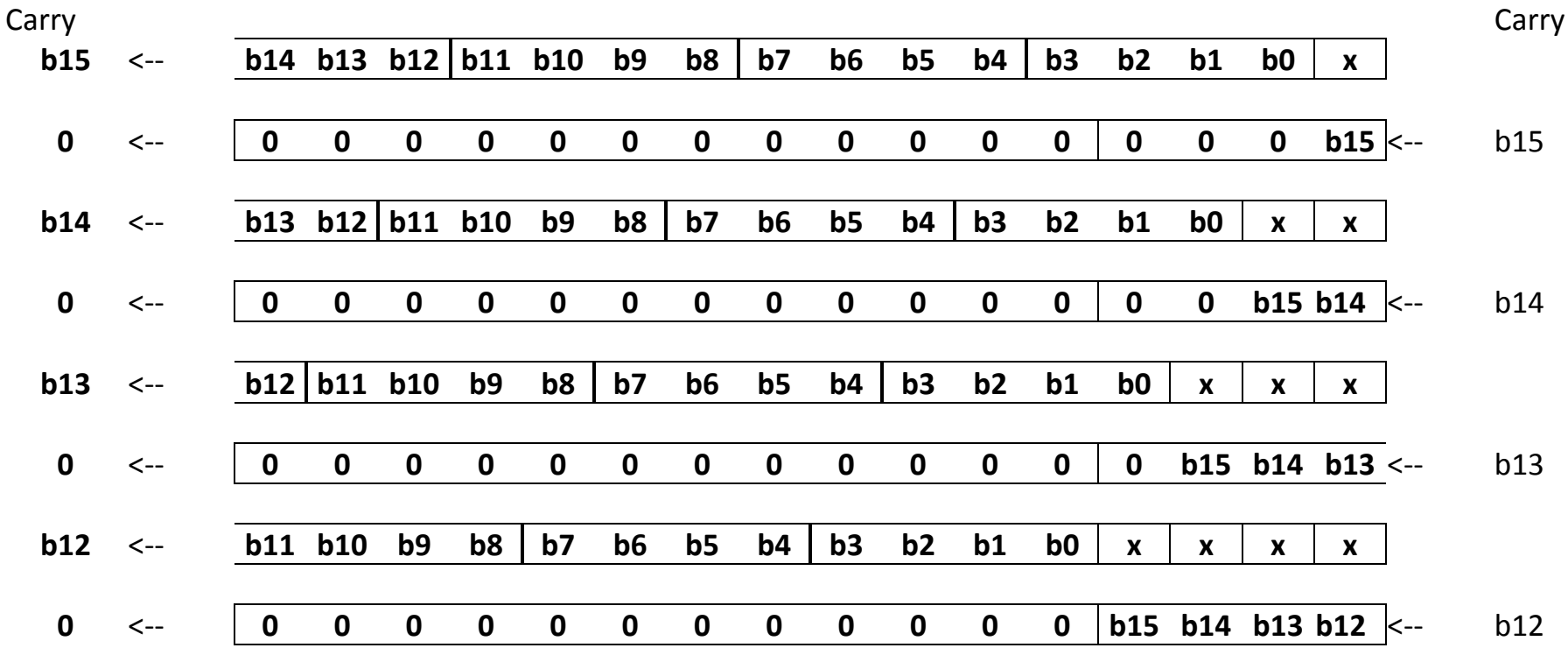
What we mean by extraction:



We want to extract only the most significant nibble and put it in one register where all other nibbles are 0, so we can use it as a number to add.

Extraction Process:
shift or rotate as you extract





Extract_Left_Nibble

```
Extract_Left_Nibble   MACRO   dato, extractor
                      mov   #0,extractor   ; initialize extractor
                      rla   dato           ; b15 to carry
                      rlc   extractor  ;  b15 least significant bit
                      rla   dato           ; b14 to carry
                      rlc   extractor  ;  b14 lsb  (   b15-b14)
                      rla   dato           ; b13 to carry
                      rlc   extractor  ;  b13 lsb  (   b15-b14-b13)
                      rla   dato           ; b12 to carry
                rlc   extractor  ;  b12 lsb  (   b15-b14-b13-b12)
                      ENDM
```

Note: no labels have been used within macro. If included, you must define local variables or else there will be in error in compiling. See Section 4.6

Multiply by ten (A)

- Ah = 1010b= 2^3 + 2
- Thus: N*A = (2^3)*N + 2*N
- Pseudocode:
  - Multiply N by 2 → P = 2P
  - Store previous result → Store P
  - Multiply again by 2 → P=2P (4P)
  - Multiply again by 2 → P=2P (8P)
  - Add stored number → 10*P

Mult_by_ten: MACRO

```
Mult_by_ten   MACRO   dato
              rla   dato  ; 2*Dato
              push dato  ; store  2*Dato
              rla    dato  ; 4*Dato
              rla    dato  ; 8*Dato
              add  @SP+, dato  ; 10*Dato
                              ; and restore TOS
              ENDM
```

Main process

- Extract N3
- Conversion = N3 ; Initialize conversion
- Multiply Conversion by Ten ; N3*A
- Extract N2
- Conversion = Conversion + N2 ; N3*A + N2
- Multiply Conversion by Ten ; (N3*A+N2)*A
- Extract N1
- Conversion = Conversion + N1 ; (N3*A + N2)*A+N1
- Multiply Conversion by Ten ;
- Extract N0
- Conversion = Conversion + N0 ; **Conversion Finished**.

Remaks upon Main Process

- Extraction macro also realizes a shift within the original datum. For this reason, extraction can be used successively.
- We can make a small change in the process and see more clearly the repetitive process:

1. Initialize conversion=0 and R=dato (N3 N2 N1 N0)
2. Repeat 4 times: (for x = 3 to 0, step -1)
   2.1 Multiply Conversion by Ten (conversion = conversion *A)
   2.2 Extract left nibble of R (Extract Nx)
   2.3 Add nibble to Conversion (conversion = conversion + Nx)

Main Code: (see how comments link code to process development)

```
Extract_Left_Nibble  R4, R6   ; Extract N3
 mov  R6, R10                 ; Initialize conversion
 Mult_by_ten  R10             ;  P*A = N3*A
 Extract_Left_Nibble  R4, R6   ; Extract N2
 add  R6, R10                  ; update conversion
                               ; N3*A+N2
 Mult_by_ten  R10              ;  P=P*A
 Extract_Left_Nibble  R4, R6   ; Extract N1
  add  R6, R10                  ; update conversion
                               ; (N3*A+N2)*A+N1
  Extract_Left_Nibble  R4, R6   ; Extract N0
  add  R6, R10                  ; update and finish  conversion
                                ; ((N3*A+N2)*A+N1)*A+N0
```

Complete code to put in assembler and test conversion of 3896

```
#include “msp430.h”
        (Include the macro definitions here; label on first column)
              ORG 0xF800    ; program memory add.
MyDATA      DW  3896h, 3896   ;  example and control
RESET       mov  #280h,SP ; init SP
StopWDT    mov  #WDTPWD+WDTHOLD,&WDTCTL
              mov &MyDATA, R4    ;  example to R4
      ( TYPE  HERE THE Main CODE;  RESULT IN R10)
              jmp  $     ; breakpoint
              ORG  0xFFFE        ; to put reset vector
               DW   RESET
```

Assignment

1. Write the code and run the assembler (do not run program)
2. After the code is assembled, open the memory window (view menu) and write down, in word size, the data at memory locations 0xF800 and 0xF802.
    1. The first one is your decimal number, the second one is the equivalent in hex system.
3. If not yet open, open the disassembly windows (view menu) and start scrolling down.
    1. ¿What was the actual effect of the macro? Did it save lines in the actual source code loaded onto the microcontroller?
    2. Do you justify the use of macros? Explain your answer
4. On the disassembly window, and on the source code, the “green” instruction is the one to be executed.
    1. In what address is the programming starting? What is the value of PC before starting to execute the program.
    2. Start executing one instruction at a time. You will notice that at a certain point the “green” line is advancing in the disassembly window, but not in the source file. Why?

Assignment (cont)

5. Run the program.
    1. Did it run correctly? How do you know that?
    2. Try another example
6. Modify the program so you may convert decimal numbers greater than 9999. (For example 897658 and 98760123
    1. Explain your algorithm
    2. Explain how you are going to verify the correctness of your code when you run it
    3. Test your program