Embedded Systems

Dr. Sajesh Kumar U.

Module IV- Architectural Support for HLL

- Abstractions in software design
- Assembly level languages
- High level languages

Data Types

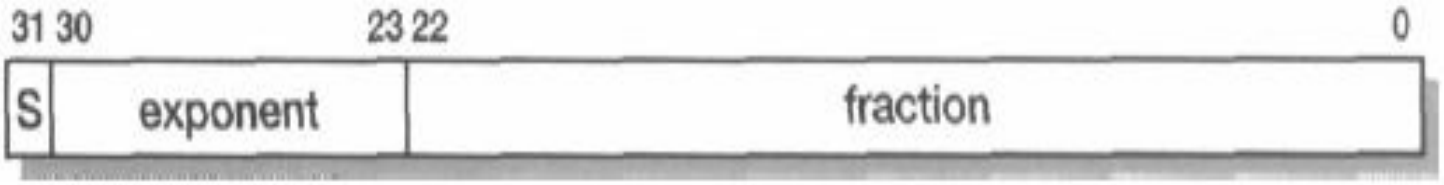
- Signed and unsigned **characters** of at least eight bits.
- Signed and unsigned **short integers** of at least 16 bits.
- Signed and unsigned **integers** of at least 16 bits.
- Signed and unsigned **long integers** of at least 32 bits.
- **Floating-point, double** and **long double** floating-point numbers.
- **Enumerated** types.
- **Bitfields.**

Data Types

- **Arrays** of several objects of the same type.
- **Functions** which return an object of a given type.
- **Structures** containing a sequence of objects of various types.
- **Pointers** (which are usually machine addresses) to objects of a given type.
- **Unions** which allow objects of different types to occupy the same space at different times.

Single Precision floating point types

Single precision



$$1995 = 1.111100101\ \text{Ix}2^{10}$$

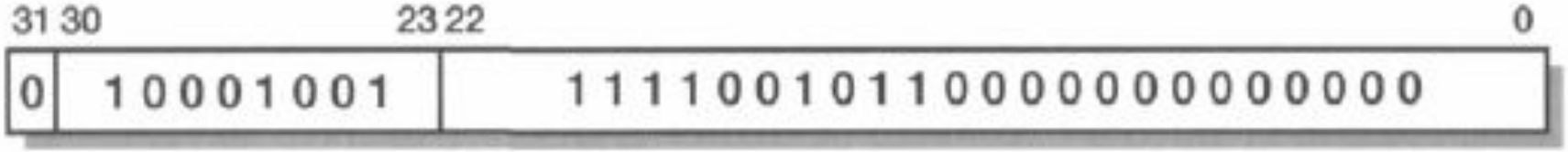
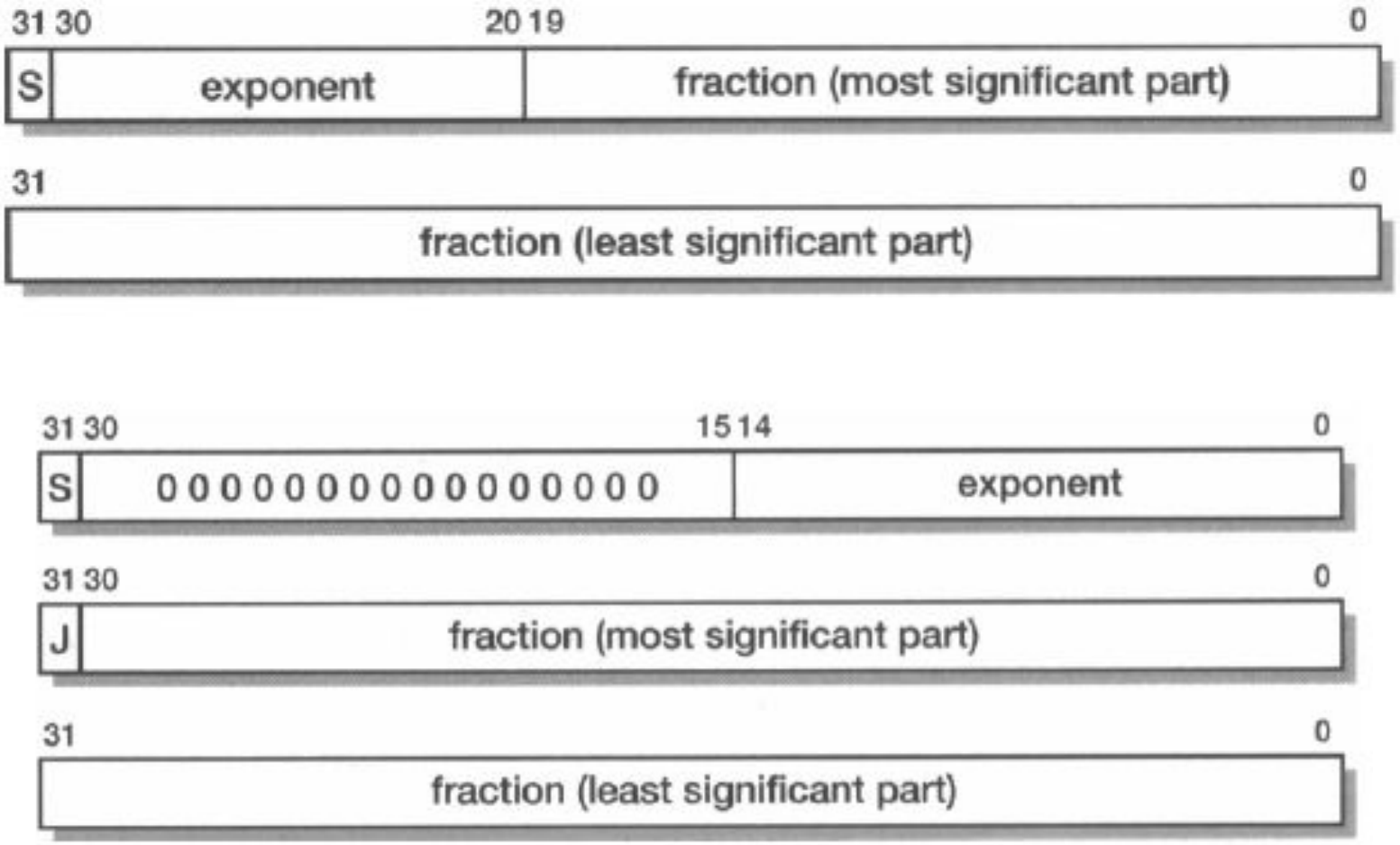
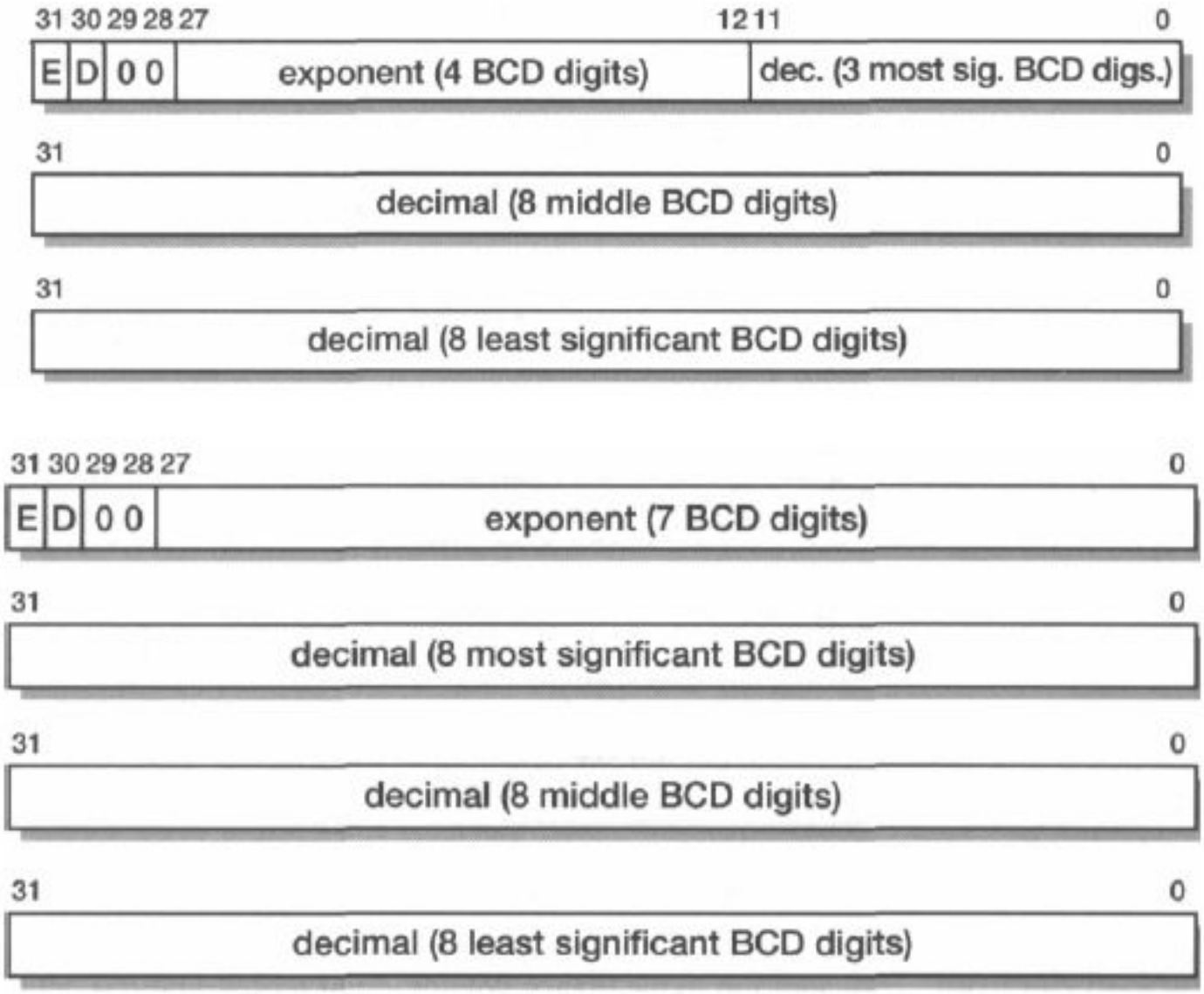


Figure 6.2 IEEE 754 single precision representation of '1995'.

Double Precision and Double extended Precision



Packed Decimal Floating Point



ARM Floating Point Architecture

- Set of floating point instructions in coprocessor instruction space.
- Either entirely implemented in software using undefined instruction trap
- Subset can be handled in hardware by coprocessor FPA10

ARM Floating Point Architecture

- Floating point system uses2 coprocessors
- Eight 80 bit registers in coprocessors
- Floating point status register
- Floating point control register

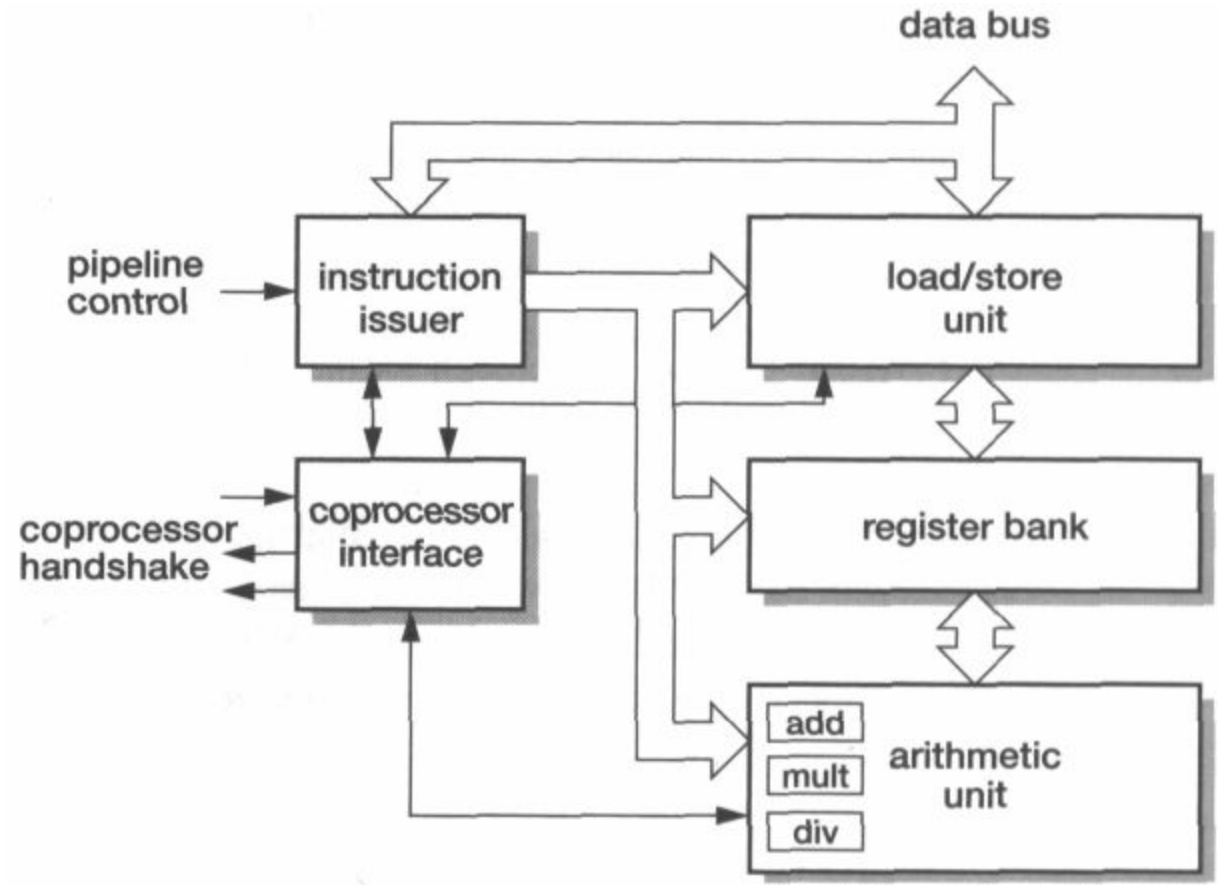
Floating Point Instructions

- Load and store floating point instructions---data size specifier
- Load and store multiple floating point instructions
- Floating point data operations---add multiply subtract divide power, remainder, transcendental functions
- Floating point register transfers

Floating point Organization

- Floating point instruction is executed as soon as it is available in instruction pipeline.
- Transferred only after ARM handshake
- When a process running the FPA is switched out FPA is turned off.
- When another process uses FPA, it will trap and the trap code will save the FPA state and FPA is turned on.

FPA10 Organization



Pointer arithmetic

- As an argument passed through a register.
- As an argument passed on the stack
- As a constant in the procedure's literal pool.
- As a local variable.
- As a global variable.

Pointers

```
Int     *p;
P = P+1;
```

```
int i = 4 ;
p = p + i;
```

If p is held in r0 and i in r1, the change to p may be compiled as:

```
ADD r0, r0, r1, LSL #2 ; scale r1 to int
```

Array representation

```
Int a[10];
```

Conditional Statements

If (a>b) c=a; else c=b;

```
CMP r0, r1

MOVGT r2, r0

MOVLE r2, rl
```

```
        CMP r0, r1
        BLE ELSE
        MOV r2, r0
        B ENDIF
ELSE    MOV r2, rl
ENDIF
```

Switch Expression-CASE

```
switch (expression) {
case constant-expression^:
statements1;
Case constant-expressio^:
statementS2
case constant-expression^:
statements^
default: statements^
}
```

R0 contains value of expression

```
    ADR r1, JUMPTABLE
    CMP r0, #TABLEMAX
    LDRLS pc, [r1,r0,LSL #2]
    ; statement sd
    B EXIT
L1      ---- statements S1
    B EXIT
    ----
Ln ----statements Sn
EXIT    …
```

Loops

For (i=0; i<10; i++)
{a[i]=0}

```
        MOV R1, #0
        ADR R2,A[0]
        MOV R0,#0
LOOP    CMP R0, #10
        BGE EXIT
        STR R1, [R2,R0,LSL #2]
        ADD RO, RO, #1
        B LOOP
EXIT
```

While and Do While Loops

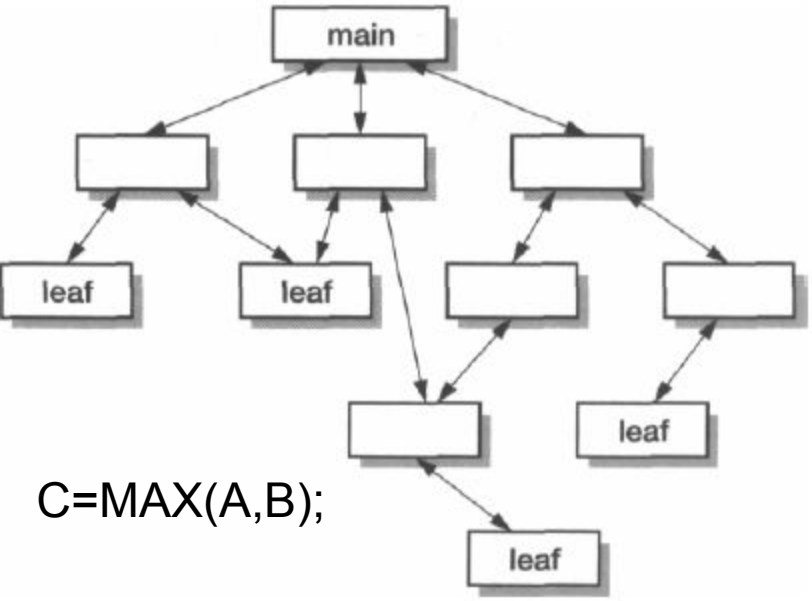
```
LOOP …   ;
EVALUATE
    BEQ EXIT
    …
    B LOOP
EXIT
```

```
    B TEST
LOOP …   ; LOOP BODY
TEST     …
    BNE LOOP
EXIT
```

```
LOOP …   ; LOOP BODY
    ….. ; EVALUATE

    BNE LOOP
EXIT
```

Functions and Procedures



C=MAX(A,B);

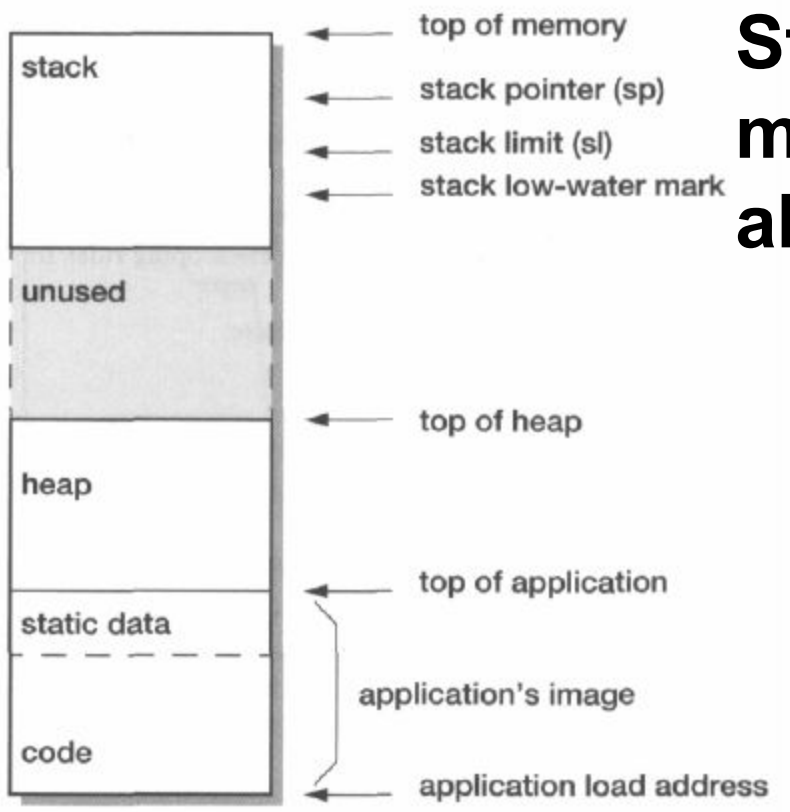
PRINTF (“Hello World\n”)

Register	APCS name	APCS role
0	a1	Argument 1 / integer result / scratch register
1	a2	Argument 2 / scratch register
2	a3	Argument 3 / scratch register
3	a4	Argument 4 / scratch register
4	v1	Register variable 1
5	v2	Register variable 2
6	v3	Register variable 3
7	v4	Register variable 4
8	v5	Register variable 5
9	sb/v6	Static base / register variable 6
10	sl/v7	Stack limit / register variable 7
11	fp	Frame pointer
12	ip	Scratch reg. / new sb in inter-link-unit calls
13	sp	Lower end of current stack frame
14	lr	Link address / scratch register
15	pc	Program counter

```
      BL  LEAF1
      …
LEAF1 …
      MOV PC, LR
```

```
      BL  LEAF2
      …
LEAF2 STMFD SP! {REGS, LR}
      …
      LDMFD SP! {REGS, PC}
```

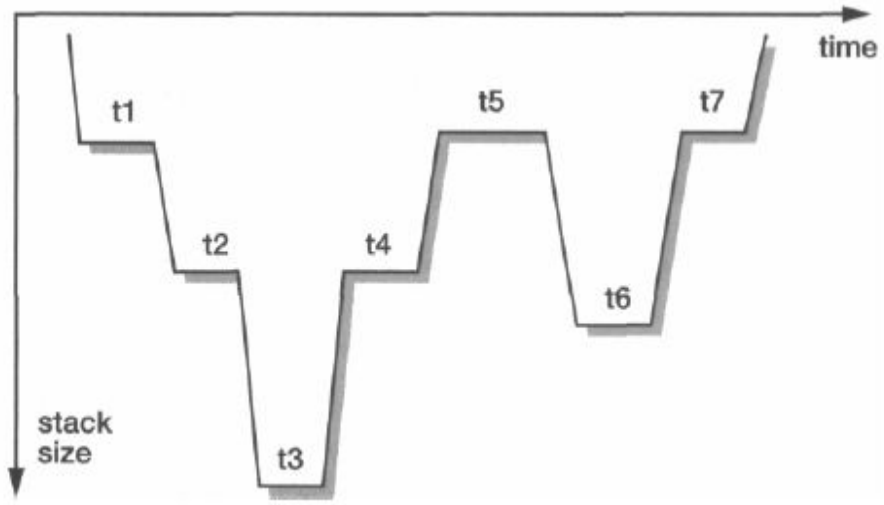
Address Space Model



Stack and Heap are allotted memory space above the code

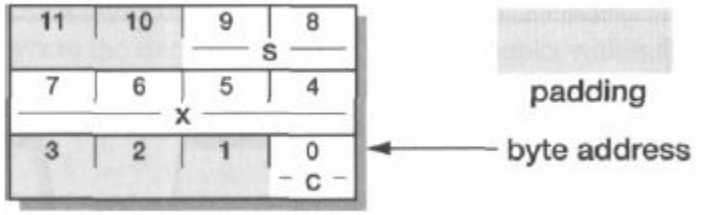
Stack Usage While Executing Functions

```
Main()   {
….
func1 {};
….
func2 {};
}
func1() {
…
func2 {};
…
}
func2()  {
…
}
```

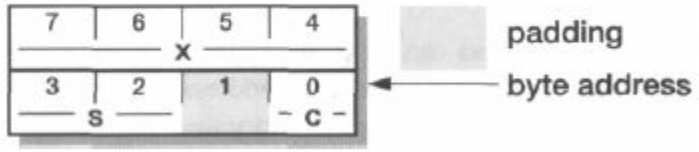


Data Alignment and Memory Efficiency

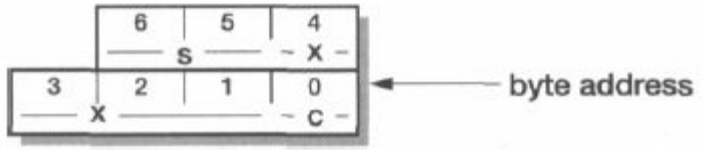
struct s1 {char c; int x; short s;} …example1



struct s2 {char c; int x; short s;} …example2



packed struct s3 {char c; int x; short s;} …example3



Run Time Environment and Library

- Division and remainder functions
- Stack limit checking functions
- Stack and heap management
- Program start up
- Program termination