EEEN301 Embedded systems

Lecture 4 2023

Computer organization

Language - continued

Procedure Calling

- Steps required
  1. Place parameters in registers X0 to X7
  2. Transfer control to procedure
  3. Acquire storage for procedure
  4. Perform procedure's operations
  5. Place result in register for caller
  6. Return to place of call (address in X30)

Procedure Call Instructions

- Procedure call: jump and link

  `BL ProcedureLabel`

  - Address of following instruction put in X30
  - Jumps to target address
- Procedure return: jump register

  `BR LR`

  - Copies LR to program counter
  - Can also be used for computed jumps
    - e.g., for case/switch statements

Leaf Procedure Example

- C code:

```
long long int leaf_example (long long int
g, long long int h, long long int i, long
long int j)
{ long long int f;
  f = (g + h) - (i + j);
  return f;
}
```

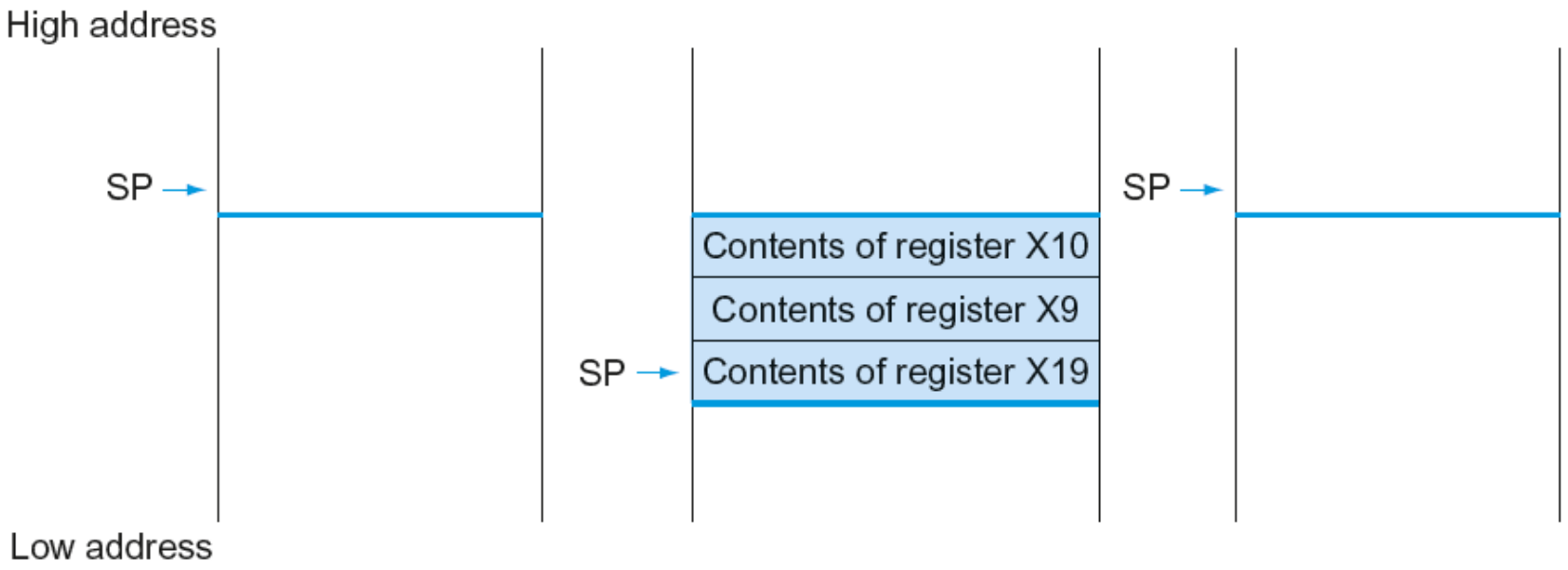
  - Arguments g, …, j in X0, …, X3
  - f in X19

Leaf Procedure Example

- LEGv8 code:

```
leaf_example:
SUBI SP,SP,#24          Save X10, X9, X19 on stack
STUR X10,[SP,#16]
STUR X9,[SP,#8]
STUR X19,[SP,#0]
ADD X9,X0,X1            X9 = g + h
ADD X10,X2,X3           X10 = i + j
SUB X19,X9,X10          f = X9 – X10
ADD X0,X19,XZR          copy f to return register
LDUR X10,[SP,#16]       Resore X10, X9, X19 from stack
LDUR X9,[SP,#8]
LDUR X19,[SP,#0]
ADDI SP,SP,#24
BR LR                   Return to caller
```

Local Data on the Stack



Register Usage

- X9 to X17: temporary registers
  - Not preserved by the callee
- X19 to X28: saved registers
  - If used, the callee saves and restores them

Non-Leaf Procedures

- Procedures that call other procedures
- For nested call, caller needs to save on the stack:
  - Its return address
  - Any arguments and temporaries needed after the call
- Restore from the stack after the call

Non-Leaf Procedure Example

- C code:

```
int fact (int n)
{
  if (n < 1) return f;
  else return n * fact(n - 1);
}
```

  - Argument n in X0
  - Result in X1

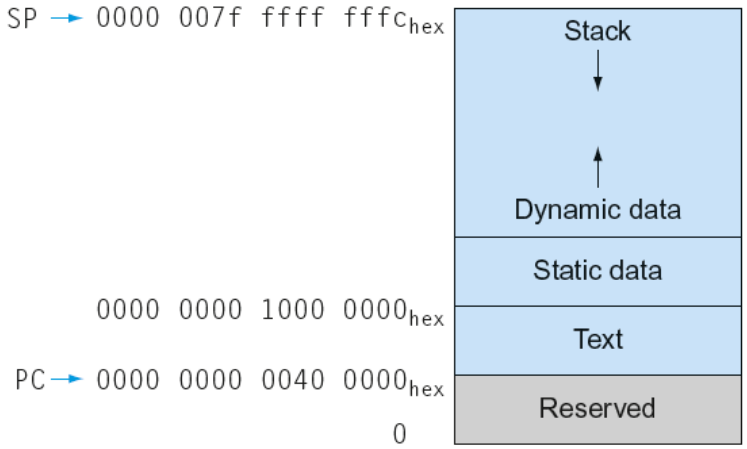
Leaf Procedure Example

- LEGv8 code:

```
   fact:
   SUBI SP,SP,#16      Save return address and n on stack
   STUR LR,[SP,#8]
   STUR X0,[SP,#0]
   SUBIS XZR,X0,#1     compare n and 1
   B.GE L1             if n >= 1, go to L1
   ADDI X1,XZR,#1      Else, set return value to 1
   ADDI SP,SP,#16      Pop stack, don't bother restoring values
   BR LR               Return
L1: SUBI X0,X0,#1      n = n - 1
   BL fact             call fact(n-1)
   LDUR X0,[SP,#0]     Restore caller's n
   LDUR LR,[SP,#8]     Restore caller's return address
   ADDI SP,SP,#16      Pop stack
   MUL X1,X0,X1        return n * fact(n-1)
   BR LR               return
```

Memory Layout

- Text: program code
- Static data: global variables
  - e.g., static variables in C, constant arrays and strings
- Dynamic data: heap
  - E.g., malloc in C, new in Java
- Stack: automatic storage



Character Data

- Byte-encoded character sets
  - ASCII: 128 characters
    - 95 graphic, 33 control
  - Latin-1: 256 characters
    - ASCII, +96 more graphic characters
- Unicode: 32-bit character set
  - Used in Java, C++ wide characters, …
  - Most of the world's alphabets, plus symbols
  - UTF-8, UTF-16: variable-length encodings

Byte/Halfword Operations

- LEGv8 byte/halfword load/store
  - Load byte:
    - `LDURB Rt, [Rn, offset]`
    - Sign extend to 32 bits in rt
  - Store byte:
    - `STURB Rt, [Rn, offset]`
    - Store just rightmost byte
  - Load halfword:
    - `LDURH Rt, [Rn, offset]`
    - Sign extend to 32 bits in rt
  - Store halfword:
    - `STURH Rt, [Rn, offset]`
    - Store just rightmost halfword

String Copy Example

- C code:
  - Null-terminated string

```
void strcpy (char x[], char y[])
{ size_t i;
  i = 0;
  while ((x[i]=y[i])!='\0')
    i += 1;
}
```

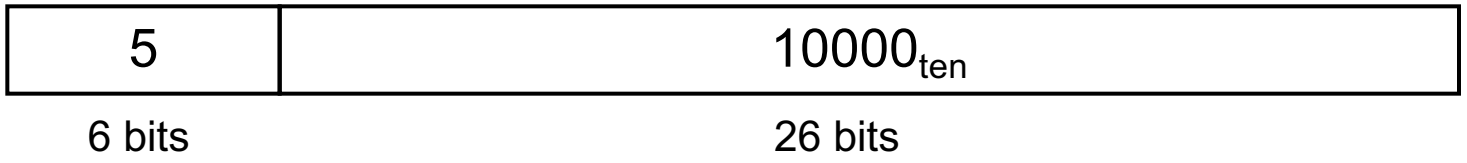
String Copy Example

- LEGv8 code:

```
strcpy:
     SUBI SP,SP,8          // push X19
     STUR X19,[SP,#0]
     ADD X19,XZR,XZR       // i=0
L1:  ADD X10,X19,X1        // X10 = addr of y[i]
     LDURB X11,[X10,#0]    // X11 = y[i]
     ADD X12,X19,X0        // X12 = addr of x[i]
     STURB X11,[X12,#0]    // x[i] = y[i]
     CBZ X11,L2            // if y[i] == 0 then exit
     ADDI X19,X19,#1       // i = i + 1
     B L1                  // next iteration of loop
L2:  LDUR X19,[SP,#0]      // restore saved $s0
     ADDI SP,SP,8          // pop 1 item from stack
     BR LR                 // and return
```

Branch Addressing

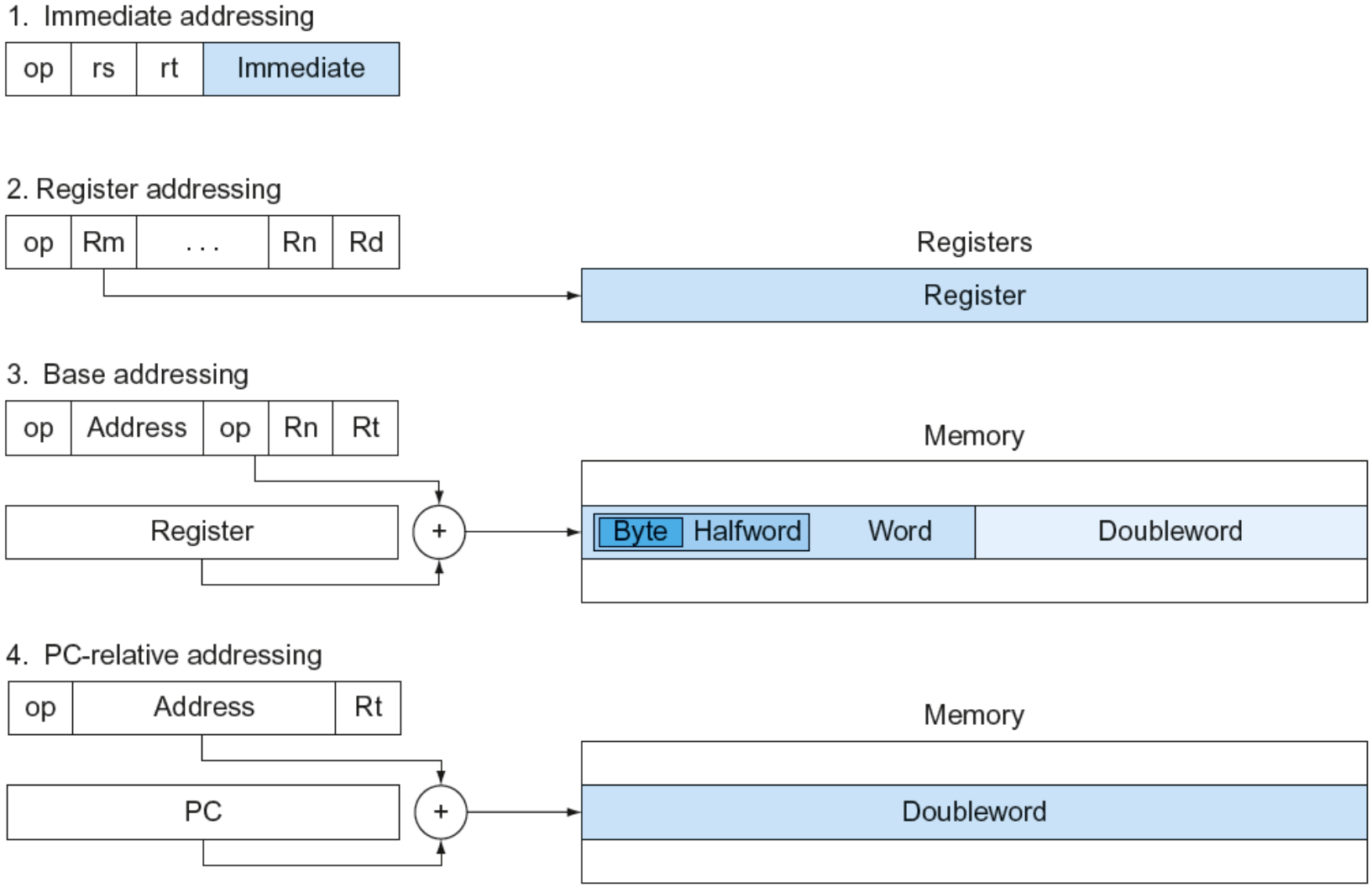
- B-type
  - `B 1000 // go to location 10000`$_{ten}$



- CB-type
  - `CBNZ X19, Exit // go to Exit if X19 != 0`

- Both addresses are PC-relative
  - Address = PC + offset (from instruction)

LEGv8 Addressing Summary



LEGv8 Encoding Summary

Name	Fields							Comments
Field size		6 to 11 bits	5 to 10 bits	5 or 4 bits	2 bits	5 bits	5 bits	All LEGv8 instructions are 32 bits long
R-format	R	opcode	Rm	shamt		Rn	Rd	Arithmetic instruction format
I-format	I	opcode	immediate			Rn	Rd	Immediate format
D-format	D	opcode	address		op2	Rn	Rt	Data transfer format
B-format	B	opcode	address					Unconditional Branch format
CB-format	CB	opcode	address				Rt	Conditional Branch format
IW-format	IW	opcode	immediate				Rd	Wide Immediate format

Synchronization

- Two processors sharing an area of memory
  - P1 writes, then P2 reads
  - Data race if P1 and P2 don’t synchronize
    - Result depends on order of accesses
- Hardware support required
  - Atomic read/write memory operation
  - No other access to the location allowed between the read and write
- Could be a single instruction
  - E.g., atomic swap of register ↔ memory
  - Or an atomic pair of instructions

Synchronization in LEGv8

- Load exclusive register: LDXR
- Store exclusive register: STXR
- To use:
  - Execute LDXR then STXR with same address
  - If there is an intervening change to the address, store fails (communicated with additional output register)
  - Only use register instruction in between

Synchronization in LEGv8

- Example 1: atomic swap (to test/set lock variable)

```
again:  LDXR X10,[X20,#0]
        STXR X23,X9,[X20] // X9 = status
        CBNZ X9, again
        ADD X23,XZR,X10 // X23 = loaded value
```

- Example 2: lock (to lock a section using a key)

```
        ADDI X11,XZR,#1       // copy locked value
again:  LDXR X10,[X20,#0]     // read lock
        CBNZ X10, again       // check if it is 0 yet
        STXR X11, X9, [X20]   // attempt to store
        BNEZ X9,again         // branch if fails
```

- Unlock:

```
        STUR XZR, [X20,#0]    // free lock
```

ARM & MIPS Similarities

- ARM: the most popular embedded core
- Similar basic set of instructions to MIPS

	ARM	MIPS
Date announced	1985	1985
Instruction size	32 bits	32 bits
Address space	32-bit flat	32-bit flat
Data alignment	Aligned	Aligned
Data addressing modes	9	3
Registers	15 × 32-bit	31 × 32-bit
Input/output	Memory mapped	Memory mapped

Instruction Encoding