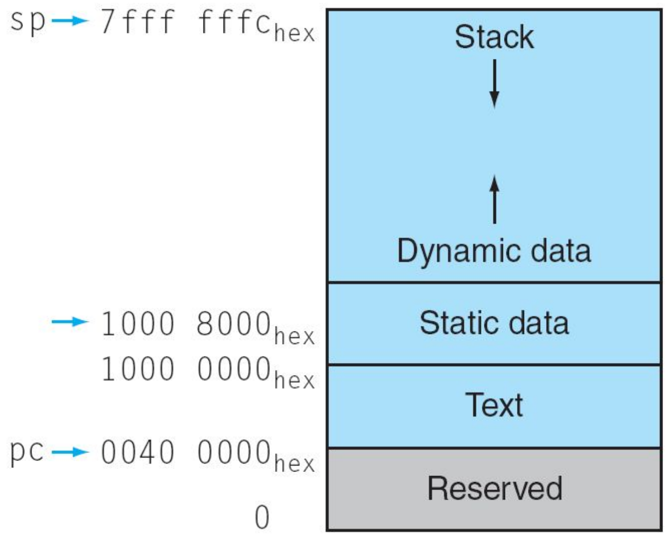
Memory Layout

- 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



Supporting Procedures in Computer Hardware

Procedure Calling

- Steps required
  1. Place parameters in registers
  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

ARM register conventions

Name	Register number	Usage	Preserved on call?
`a1-a2`	0–1	Argument / return result / scratch register	no
`a3-a4`	2–3	Argument / scratch register	no
`v1-v8`	4–11	Variables for local routine	yes
`ip`	12	Intra-procedure-call scratch register	no
`sp`	13	Stack pointer	yes
`lr`	14	Link Register (Return address)	yes
`pc`	15	Program Counter	n.a.

Procedure Call Instructions

- Procedure call: Branch and link

  ```
  BL ProcedureAddress
  ```

  - Address of following instruction put in ***lr***
  - Jumps to target address
- Procedure return:

  ```
  MOV pc,lr
  ```

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

Leaf Procedure Example

- C code:

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

  - *Arguments g, …, j in r0, …, r3*
  - *f in r4 (hence, need to save r4 on stack)*
  - *Result in r0*

Leaf Procedure Example

- ARM code:

```
leaf_example:
    SUB sp, sp, #12        Make room for 3 items
    STR r6,[sp,#8]
    STR r5,[sp,#4]         Save r4,r5,r6 on stack
    STR r4,[sp,#0]
    ADD r5,r0,r1           r5 = (g+h), r6= (i+j)
    ADD r6,r2,r3           Result in r4
    SUB r4,r5,r6
     MOV r0,r4             Result moved to return
                           value register r0.
     LDR r4, [sp,#0]
     LDR r5, [sp,#4]       Restore r4,r5,r6
     LDR r6, [sp,#8]
    ADD sp,sp,#12
    MOV pc,lr              Return
```

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

The template (used in the lab)

```
main:
    @ stack handling, will discuss later
    @ push (store) lr to the stack
    sub sp, sp, #4
    str lr, [sp, #0]
    @ Write YOUR CODE HERE
    @ ---------------------

    @ ---------------------
    @ stack handling (pop lr) and return
    ldr lr, [sp, #0]
    add sp, sp, #4
    mov pc, lr
```