Fundamentals of Computer Architecture

ARM – Arrays and Function Calls

Teacher: Lobar Asretdinova

Programming Building Blocks

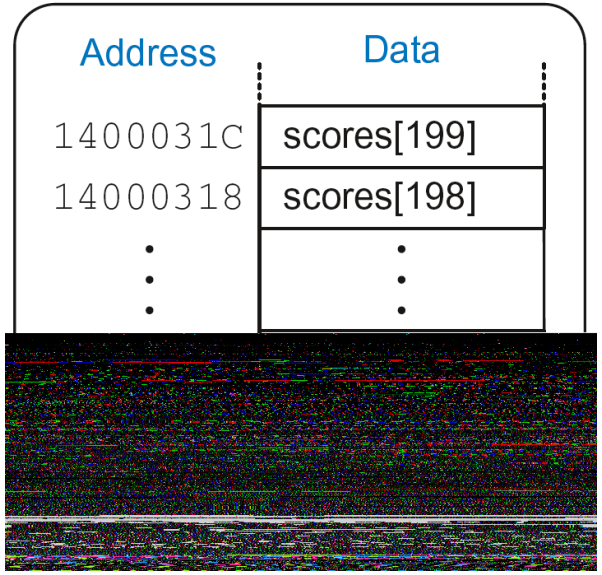
- Data-processing Instructions
- Conditional Execution
- Branches
- High-level Constructs:
  - if/else statements
  - for loops
  - while loops
  - arrays
  - function calls

Arrays

- Access large amounts of similar data
  - Index: access to each element
  - Size: number of elements

Arrays

- 5-element array
  - **Base address** = 0x14000000 (address of first element, scores[0])
  - Array elements accessed relative to base address



Accessing Arrays

C Code

```
int array[5];
array[0] = array[0] * 8;
array[1] = array[1] * 8;
```

ARM Assembly Code

```
; X0 = array base address
```

Accessing Arrays

C Code

```
int array[5];
array[0] = array[0] * 8;
array[1] = array[1] * 8;
```

ARM Assembly Code

```
; X0 = array base address
  MOV X0, #0x60000000          ; X0 = 0x60000000

  LDR X1, [X0]                 ; X1 = array[0]
  LSL X1, X1, 3                ; X1 = X1 << 3 = X1*8
  STR X1, [X0]                 ; array[0] = X1

  LDR X1, [X0, #8]             ; X1 = array[1]
  LSL X1, X1, 3                ; X1 = X1 << 3 = X1*8
  STR X1, [X0, #8]             ; array[1] = X1
```

Arrays using for Loops

C Code

```
int array[200];
int i;

for (i=199; i >= 0; i = i - 1)
     array[i] = array[i] * 8;
```

ARM Assembly Code

```
; X0 = array base address, R1 = i
```

Arrays using for Loops

C Code

```
int array[200];
int i;

for (i=199; i >= 0; i = i - 1)
     array[i] = array[i] * 8;
```

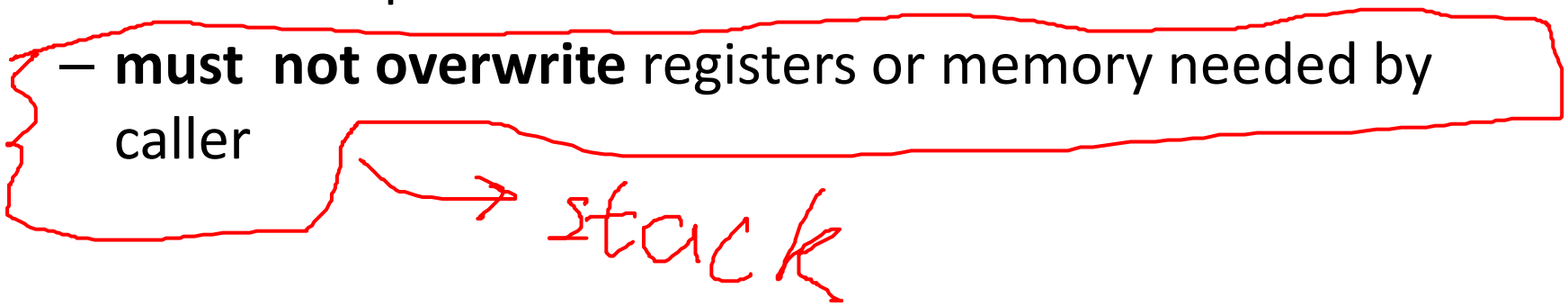
ARM Assembly Code

```
// X0 = array base address, X1 = i
  MOV X0, 0x60000000
  MOV X1, #199

FOR:
  LDR  X2, [X0, X1, LSL #3]      // X2 = array(i)
  LSL  X2, X2, #3                // X2 = X2<<3 = X3*8
  STR  X2, [X0, X1, LSL #3]      // array(i) = X2
  SUBS X1, X1, #1                // i = i - 1
                                 // and set flags
  BPL  FOR                       // if (i>=0) repeat loop
```

Function Conventions

- **Caller:**
  - passes **arguments** to callee
  - jumps to callee
- **Callee:**
  - **performs** the function
  - **returns** result to caller
  - **returns** to point of call
  - **must not overwrite** registers or memory needed by caller



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

ARM Function Conventions

- **Call Function:** branch and link

  `BL`

- **Return** from function: move the link register to PC: `MOV PC, LR`

- **Arguments:** `X0-X7`
- **Return value:** `X0`

Function Calls

C Code

```
int main() {
  simple();
  a = b + c;
}
```

```
void simple() {
  return;
}
```

ARM Assembly Code

```
0x00000200 MAIN     BL   SIMPLE
0x00000204          ADD X4, X5, X6
...

0x00401020 SIMPLE   MOV PC, LR
```

Function Calls

C Code

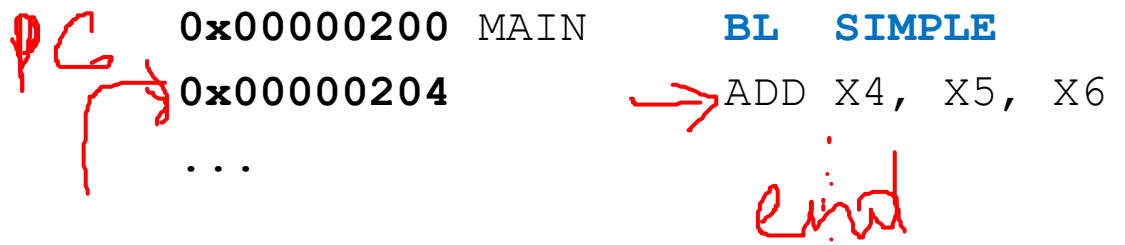
```
int main() {
  simple();
  a = b + c;
}

void simple() {
  return;
}
```

ARM Assembly Code

```
0x00000200 MAIN     BL   SIMPLE
0x00000204          ADD X4, X5, X6
...

0x00401020 SIMPLE   MOV PC, LR
```



BL branches to SIMPLE
LR = PC + 4 = 0x00000204

MOV PC, LR makes PC = LR
(the next instruction executed is at 0x00000200)

Input Arguments and Return Value

C Code

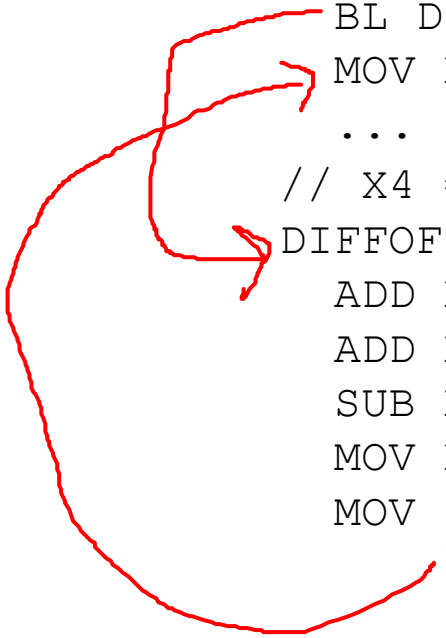
```
int main()
{
  int y;
  ...
  y = diffofsums(2, 3, 4, 5);  // 4 arguments
  ...
}

int diffofsums(int f, int g, int h, int i)
{
  int result;
  result = (f + g) - (h + i);
  return result;                // return value
}
```

Input Arguments and Return Value

ARM Assembly Code

```
// X4 = y
MAIN
   ...
   MOV X0, #2          ; argument 0 = 2
   MOV X1, #3          ; argument 1 = 3
   MOV X2, #4          ; argument 2 = 4
   MOV X3, #5          ; argument 3 = 5
   BL DIFFOFSUMS       ; call function
   MOV X4, X0          ; y = returned value
   ...
// X4 = result
DIFFOFSUMS
   ADD X8, X0, X1      ; X8 = f + g
   ADD X9, X2, X3      ; X9 = h + i
   SUB X4, X8, X9      ; result = (f + g) - (h + i)
   MOV X0, X4          ; put return value in X0
   MOV PC, LR          ; return to caller
```



Input Arguments and Return Value

ARM Assembly Code

```
// X4 = result
DIFFOFSUMS
  ADD x8, X0, X1      ; X8 = f + g
  ADD x9, X2, X3      ; X9 = h + i
  SUB x4, X8, X9      ; result = (f + g) - (h + i)
  MOV X0, X4          ; put return value in X0
  MOV PC, LR          ; return to caller
```

- `diffofsums` overwrote 3 registers: `X4`, `X8`, `X9`
- `diffofsums` can use *stack* to temporarily store registers