CSE 30: Computer Organization and Systems Programming

Lecture 15: ARM Loops and ARM Procedure Call Standard

Diba Mirza
University of California, San Diego
While loops

while (a<0)
    a++;

do
{
    a++;
}while(a<0);
For loops

for (i=0; i<10; i++){
    a++;  
    b--;  
}
C functions

```c
main() {
    int a=10, b=20, c;
    c = sum(a, b); /* a, b, c: r0, r1, r2 */
    ...
}

/* sum function */
int sum(int x, int y) {
    return x+y;
}
```

CalleR: the calling function
CalleE: the function being called
main() {
    int a=10, b=20, c;
    c = sum(a, b);
    ...
}

/* sum function */
int sum(int x, int y) {
    return x+y;
}
Making the function call

```c
... sum(a, b); ... /* a, b: r4, r5 */
}
int sum(int x, int y) {
    return x + y;
}
```

Is there something wrong with using the simple branch instruction?

A. Yes
B. No

address
1000 ... Is there something wrong with using the A. Yes
1004 ... simple branch instruction? B. No
1008 ...
1012 B sum ; branch to sum
1016 return_loc: ...
1020 ...
2000 sum: ADD r0, r0, r1
2004 B return_loc
Using the branch instruction....

c=sum(a,b)
c=c+1;
.....
}
int sum(int x, int y) {
  return x+y;
}

address          Is there something wrong with using the A. Yes
1000 ...          simple branch instruction? B. No
1004 ...
1008 ...
1012 B sum ;branch to sum
1016 return_loc:...
1020 ...
2000 sum: ADD r0,r0,r1
2004 B return_loc

Reason: sum might be called by many functions, so we can’t return to a fixed place.
The calling proc to sum must be able to say “return back here” somehow.
Making a function call

... sum(a,b);... /* a,b:r4,r5 */
}
int sum(int x, int y) {
  return x+y;
}

address
1000  ...
1004  ...
1008  MOV  lr, 1016 ; lr = 1016
1012  B    sum ; branch to sum
1016  ...
1020  ...
2000  sum: ADD r0,r0,r1
2004  BX  lr ; MOV  pc,lr i.e., return
Instruction Support for Functions

Single instruction to jump and save return address: jump and link (BL)

• Before:
  1008  MOV  lr, 1016    ; lr=1016
  1012  B  sum    ; go to sum

• After:
  1008  BL  sum    # lr=1012, goto sum

Why have a BL? Make the common case fast: function calls are very common. Also, you don’t have to know where the code is loaded into memory with BL.
main() {
    int a=10, b=20, c;
    c = sum(a, b);
    c = c + 1;
    ....
}

/* sum function */
int sum(int x, int y) {
    return x + y;
}
Register Conventions

• **Register Conventions**: A set of generally accepted rules as to which registers are guaranteed to be unchanged after a procedure call (BL) and which may be changed.
Register Usage

Registered variables

Must be preserved

Arguments into function
Result(s) from function
otherwise corruptible
(Additional parameters passed on stack)

The compiler has a set of rules known as a Procedure Call Standard that determine how to pass parameters to a function (see AAPCS)

CPSR flags may be corrupted by function call. Assembler code which links with compiled code must follow the AAPCS at external interfaces

Scratch register (corruptible)

Stack Pointer
Link Register
Program Counter

- Stack base
- Stack limit if software stack checking selected
- SP should always be 8-byte (2 word) aligned
- R14 can be used as a temporary once value stacked
Example: Caller Assembly code

```assembly
main() {
    int i, j, k, m; /* i-m:r4-r7 */
    i = mult(j, k);
    m = mult(i, i) + j;
}

main:
    MOV               ; arg1 = j
    MOV               ; arg2 = k
    BL mult           ; call mult
    MOV               ; arg1 = i
    MOV               ; arg2 = i
    BL mult           ; call mult
    ; m=mult(i,i)+j
```
Example: Caller Assembly code

```c
main() {
    int i, j, k, m; /* i-m:r4-r7 */
    i = mult(j, k);
    m = mult(i, i) + j;
}

main:
    MOV r0, r5 ; arg1 = j
    MOV r1, r6 ; arg2 = k
    BL mult ; call mult
    MOV r1, r0 ; arg2 = i
    BL mult ; call mult
    ADD r7, r0, r5 ; m = mult(i, i) + j
```
int mult (int mcand, int mlier) {
    int product = 0;
    while (mlier > 0) {
        product += mcand;
        mlier -= 1;
    }
    return product;
}

mult:  MOV r4,#0; prod=0
Loop:  CMP r1, #0 ; mlier == 0?
       BGT Fin   ; if mlier > 0 goto Fin
       ADD r4,r4,r1 ; product += mcand
       ADD r1,r1,#-1 ; mlier -= 1
       B Loop     ; goto Loop
Fin:   BX lr   ; return

How many errors are there in the ARM code given below?
A. Zero
B. One
C. Two
D. Three
E. More than three
int mult (int mcand, int mlier){
    int product = 0;
    while (mlier > 0) {
        product += mcand;
        mlier -= 1; }
    return product;
}

How to save the “saved registers”?

mult:
    MOV r4,#0 ; prod=0
Loop: CMP r1, #0 ; mlier == 0?
    BLE Fin ; if mlier <= 0 goto Fin
    ADD r4,r4,r1 ; product += mcand
    ADD r1,r1,#-1 ; mlier -= 1
    B Loop ; goto Loop

Fin: BX lr ; return