Tutorial 5.2

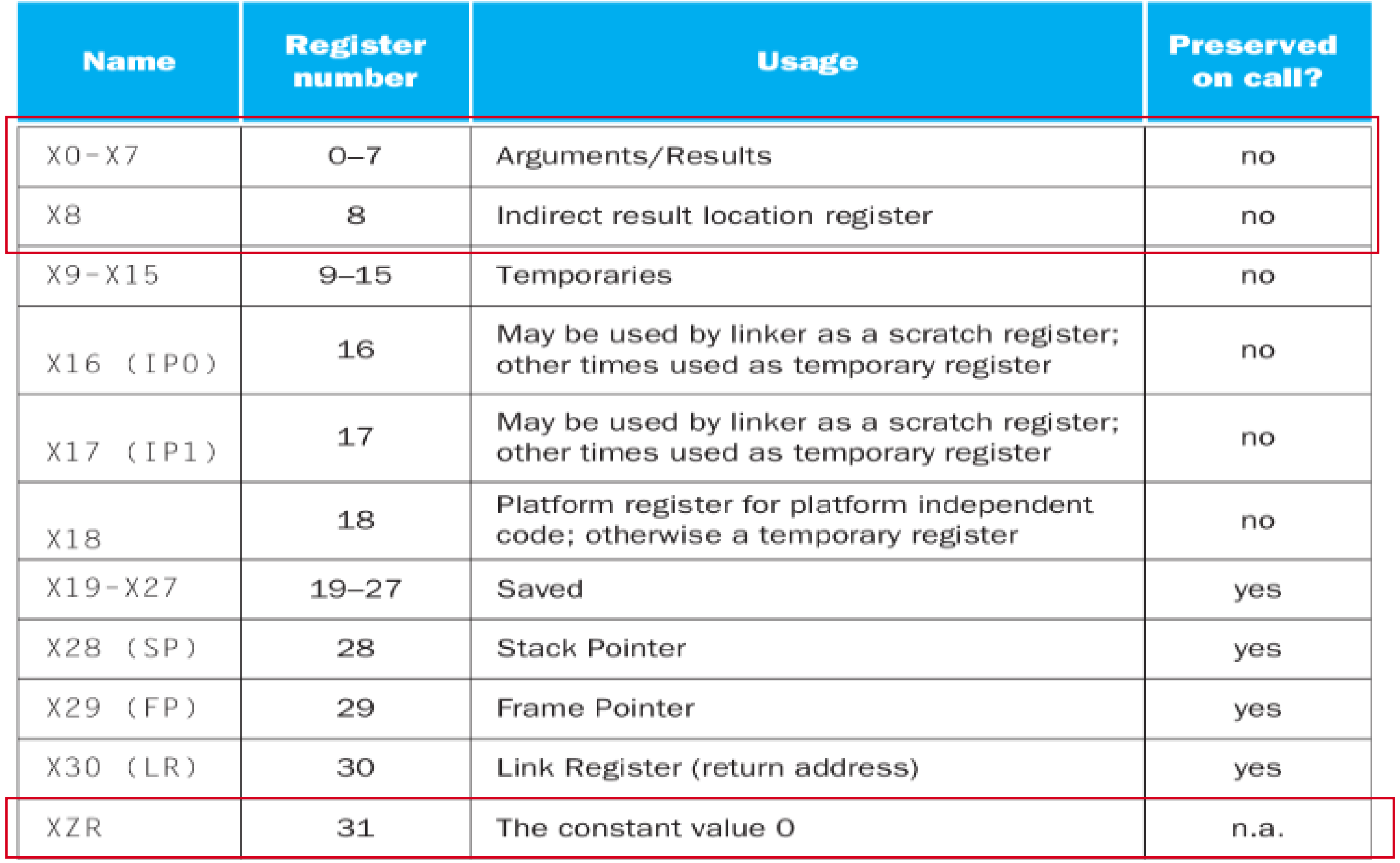
subroutines incorporating local variables & structures

Lei Wang

lei.wang2@ucalgary.ca

UNIVERSITY OF CALGARY

Quick overview of registers



Name	Register number	Usage	Preserved on call?
X0-X7	0–7	Arguments/Results	no
X8	8	Indirect result location register	no
X9-X15	9–15	Temporaries	no
X16 (IP0)	16	May be used by linker as a scratch register; other times used as temporary register	no
X17 (IP1)	17	May be used by linker as a scratch register; other times used as temporary register	no
X18	18	Platform register for platform independent code; otherwise a temporary register	no
X19-X27	19–27	Saved	yes
X28 (SP)	28	Stack Pointer	yes
X29 (FP)	29	Frame Pointer	yes
X30 (LR)	30	Link Register (return address)	yes
XZR	31	The constant value 0	n.a.

Subroutines with 9 or More Arguments

- Arguments beyond the 8th are passed on the stack
  - the calling code allocates memory at the top of the stack, and writes the ”spilled” argument values there
    - by convention, each argument is allocated 8 bytes
  - The callee reads this memory using the appropriate offset

Coding example

translate this into assembly

```
int sum(int a1, int a2, int a3, int a4, int a5,
        int a6, int a7, int a8, int a9, int a10);

int main()
{
        register int result;
        result  = sum(10, 20, 30, 40, 50, 60, 70, 80, 90, 100);
}

int sum(int a1, int a2, int a3, int a4, int a5,
                 int a6, int a7, int a8, int a9, int a10)
{
        return a1 + a2 + a3 + a4 + a5 + a6 + a7 + a8 + a9 + a10;
}
```

Coding example

```
arg9_s = 16
arg10_s = 24

sum:        stp         x29, x30, [sp, -16]!
            mov         x29. sp
            // add first 8 args
            add         w0, w0, w1
            add         w0, w0, w1
            add         w0, w0, w3
            add         w0, w0, w4
            add         w0, w0, w5
            add         w0, w0, w6
            add         w0, w0, w7
            // add 9th and 10th args
            ldr         w9, [x29, arg9_s]
            add         w0, w0, w9
            ldr         w9, [x29, arg10_s]
            add         w0, w0, w9

            ldp         x29, x30, [sp], 16
            ret
```

```
define(result_r, w19)

            spilled_mem_size = 16
            alloc = -sppiled_mem_size & -16
            dealloc = -alloc

            .global main
main:       stp         x29, x30, [sp, -16]!
            mov         x29, sp
            // Set up first 8 args
            mov         w0, 10
            mov         w1, 20
            mov         w2, 30
            mov         w3, 40
            mov         w4, 50
            mov         w5, 60
            mov         w6, 70
            mov         w7, 80
            // Allocate memory for args 9 and 10
            add         sp, sp, alloc
            //write spilled args to top of stack
            mov         w9, 90
            str         w9, [sp, 0]
            mov         w9, 100
            str         w9, [sp, 8]

            bl          sum
            mov         result_r, w0

            add         sp, sp, alloc
            ldp         x29, x30, [sp], 16
            ret
```

Structure

- In assembly, a struct is defined as a group of offsets, relative to the base address for an instance of the struct.
- Usually a struct is too big to return in x0 or w0
- The calling code provides memory on the stack to store the return result
    - the address of this memory is put into x8 prior to the function call
        - x8 is the "indirect result location register"
    - the called subroutine writes to memory at this address, using x8 as a pointer to it

Example of structure

```c
struct mystruct {
        int i;
        int j;
};

struct mystruct init()
{
        struct mystruct lvar;
        lvar.i = 0;
        lvar.j = 0;
        return lvar;
}

int main()
{
        struct mystruct a;
        a = init() ;
        struct mystruct b;
        b = init() ;
}
```

```
// function: mystruct_init
define(lvar_base_r, x9)
            lvar_size = 8
            alloc = -(16 + lvar_size) & -16
            dealloc = -alloc
            lvar_s = 16

init:       stp         x29, x30, [sp, alloc]!
            mov         x29, sp
            // calculate lvar struct base address
            add         lvar_base_r, x29, lvar_s
            // initialize i and j
            str         wzr, [lvar_base_r, mystruct_i]  // lvar.i = 0
            str         wzr, [lvar_base_r, mystruct_j]  // lvar.j = 0
            // set return value in main:
            ldr         w10, [lvar_base_r, mystruct_i]
            str         w10, [x8, mystruct_i]
            ldr         w10, [lvar_base_r, mystruct_j]
            str         w10, [x8, mystruct_j]
```

instead of using w0-w7 for storing returning value, use stack memory

```
            // return
            ldp x29, x30, [sp], dealloc
            ret
// function: main
            mystruct_i = 0
            mystruct_j = 4
            mystruct_size = 8
            alloc = -(16 + 2 * mystruct_size) & -16
            dealloc = -alloc
            a_s = 16
            b_s = a_s + mystruct_size

            .global main
main:       stp         x29, x30, [sp, alloc]!
            mov         x29, sp
            // set struct a
            add         x8, x29, a_s
            bl          init
            // set struct b
            add         x8, x29, b_s
            bl          init
```

how to see the value of structure a and b?

```
done:       ldp         x29, x30, [sp], dealloc
            ret
```

Use gdb to see the value of structure

- same as array
- x/nd $fp+a_s/b_s

Exercise

- write a function to change the value of struct
- write a function to print the value of struct