Lab 4

61C Summer 2023

Calling Conventions

Caller saved	Callee saved
ra t regs a regs	sp s regs

Assume these change after a function call (If caller cares about these values then they have to save)

Assume these don't change after a function call (Callee has to explicitly save)

#	Name	Description	#	Name	Desc
x0	zero	Constant 0	x16	a6	*Args*
x1	ra	*Return Address*	x17	a7	
x2	sp	Stack Pointer	x18	s2	Saved Registers
x3	gp	Global Pointer	x19	s3	
x4	tp	Thread Pointer	x20	s4	
x5	t0	*Temporary Registers*	x21	s5	
x6	t1		x22	s6	
x7	t2		x23	s7	
x8	s0	Saved Registers	x24	s8	
x9	s1		x25	s9	
x10	a0	*Function Arguments or Return Values*	x26	s10	
x11	a1		x27	s11	
x12	a2	*Function Arguments*	x28	t3	*Temporaries*
x13	a3		x29	t4	
x14	a4		x30	t5	
x15	a5		x31	t6	
Caller saved registers					
Callee saved registers (except x0, gp, tp)					

Storing on the stack

- To save bits held by the register, we can decrement the stack pointer(stack grows downwards) and store them on the stack
- Ex.

```
addi sp sp -4
sw s0 0(sp)
```

Makes space on the stack for 4 bytes (1 word)

Stores s0 at the very bottom of the stack where sp is pointing

Storing on the stack

- To save bits held by the register, we can decrement the stack pointer(stack grows downwards) and store them on the stack
- Ex.

```
addi sp sp -4
sw s0 0(sp)
```

- To get bits stored on the stack and also restore the stack pointer, we can load values with offset from sp and decrement sp
- Ex.

```
lw s0 0(sp)
addi sp sp 4
```

Loading s0 from stack

Incrementing the stack pointer

CC Example

```
func1: # modifies a0, t0, and s0. ra points to the `main` function.
    # Checkpoint 1: Prologue
    # Some block of code using a0, t0, and s0
    # Checkpoint 2: What do you need to do before you call another function?
    # input argument at a0, return value at a0
    jal ra, func2  # call func2
    # Checkpoint 3: What do you need to do after a function call?
    # Some block of code using a0, t0, and s0
    # Checkpoint 4: Epilogue
    jr ra   # function return
```

Assume some function main calls func1 and func1 calls func2

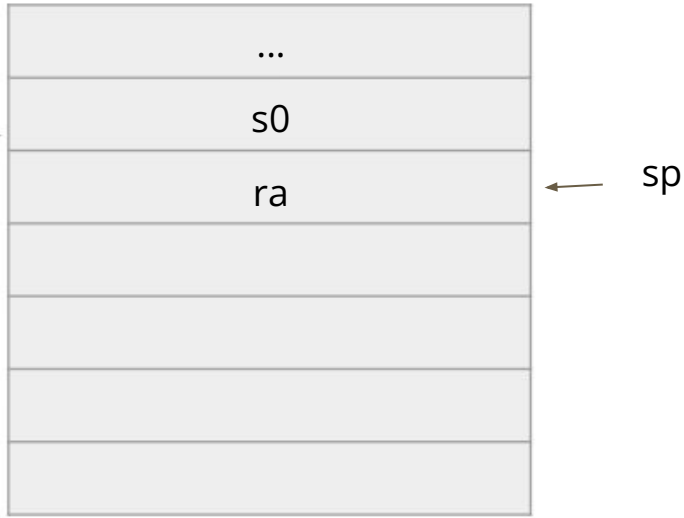
CC Example

```
func1: # modifies a0, t0, and s0. ra points to the `main` function.
    # Checkpoint 1: Prologue
    # Some block of code using a0, t0, and s0
```

What do we need to do here? (note: we modify s0, ra, a0, and t0)

CC Example

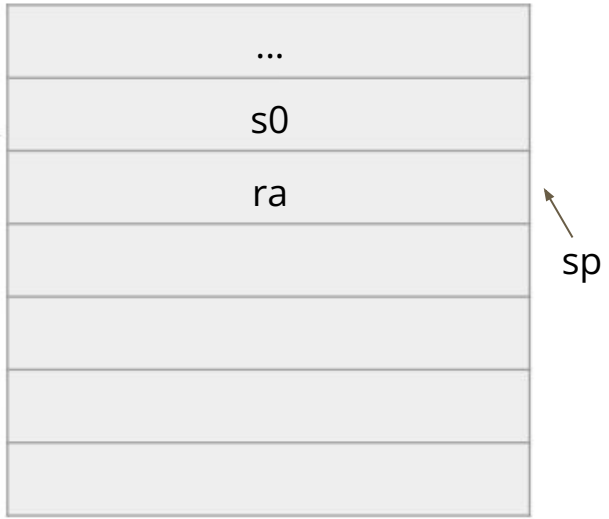
```
func1: # modifies a0, t0, and s0. ra points to the `main` function.
    # Checkpoint 1: Prologue
    addi sp sp -8   # Push the stack pointer down by 2 words (8 bytes)
    sw ra 0(sp)     # Save the return address register (ra)
    sw s0 4(sp)     # Save the saved register (s0)
    # Some block of code using a0, t0, and s0
```



CC Example

func1: # modifies a0, t0, and s0. ra points to the `main` function.

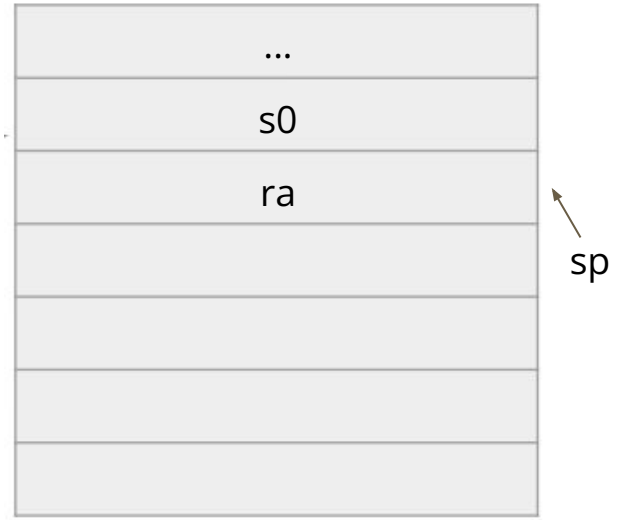
```
    # Checkpoint 1: Prologue

    # Some block of code using a0, t0, and s0

    # Checkpoint 2: What do you need to do before you call another function?

    # input argument at a0, return value at a0
```



CC Example

What should we do at checkpoint 2?

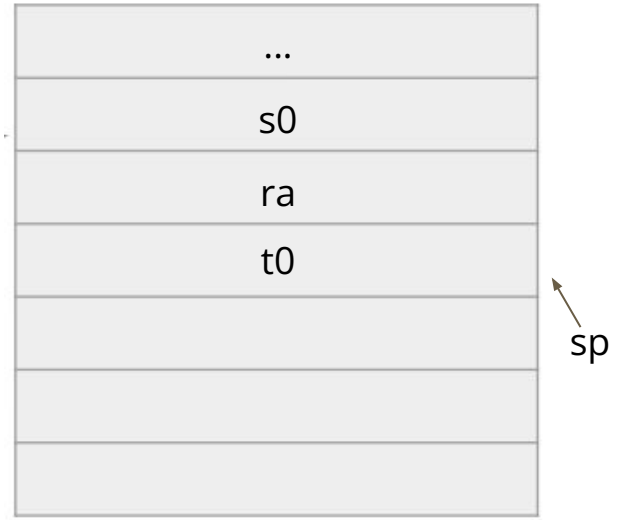
```
func1: # modifies a0, t0, and s0. ra points to the `main` function.
    # Checkpoint 1: Prologue
    # Some block of code using a0, t0, and s0
    # Checkpoint 2: What do you need to do before you call another function?
    # input argument at a0, return value at a0
    jal ra, func2  # call func2
    # Checkpoint 3: What do you need to do after a function call?
    # Some block of code using a0, t0, and s0
```



CC Example

What should we do at checkpoint 2

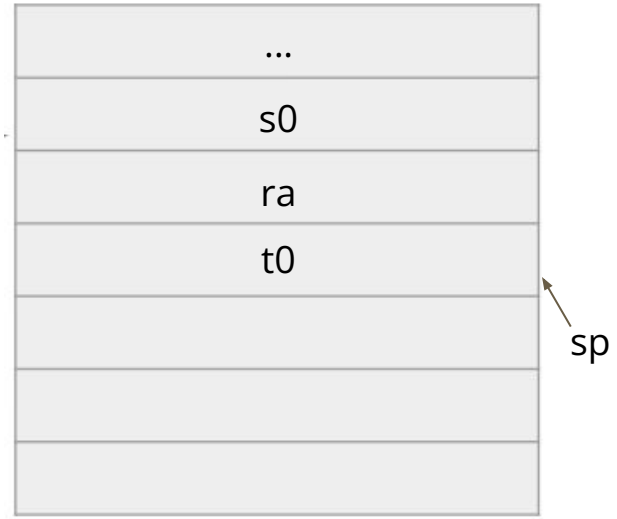
```
func1: # modifies a0, t0, and s0. ra points to the `main` function.
    # Checkpoint 1: Prologue
    # Some block of code using a0, t0, and s0
    # Checkpoint 2
    addi sp sp -4   # Push the stack pointer down by 1 word (4 bytes)
    sw t0 0(sp)     # Save the temporary register (t0)
    # input argument at a0, return value at a0
    jal ra, func2  # call func2
```



We use t0, a0, and s0. We assume s0 doesn't change. a0 is the argument so we don't need to save it

CC Example

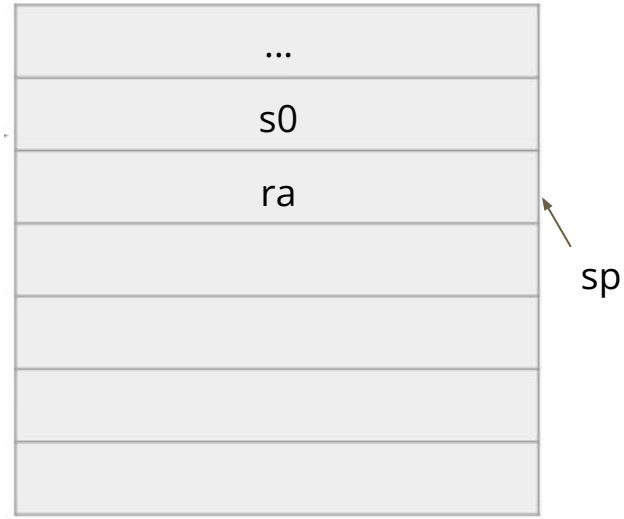
```
func1: # modifies a0, t0, and s0. ra points to the `main` function.
    # Checkpoint 1: Prologue Omitted
    # Checkpoint 2: Omitted
    jal ra, func2  # call func2
    # Checkpoint 3: What do you need to do after a function call?
    # Some block of code using a0, t0, and s0
    # Checkpoint 4: Epilogue
    jr ra   # function return
```



What do we do for ch. 3

CC Example

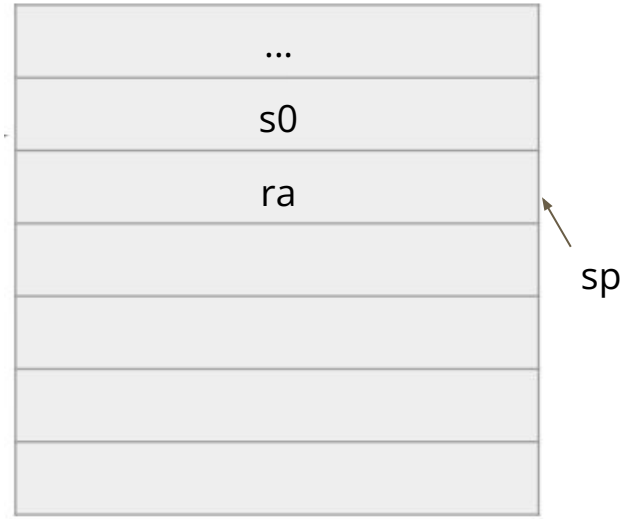
```
func1: # modifies a0, t0, and s0. ra points to the `main` function.
    # Checkpoint 1: Prologue Omitted
    # Checkpoint 2: Omitted
    jal ra, func2  # call func2
    # Checkpoint 3: What do you need to do after a function call?
    lw t0 0(sp)     # Retrieve the saved temporary register from the stack
    addi sp sp 4    # Return the stack pointer up by 1 word (4 bytes)
```



We get t0 back as we need it for the instructions that follow

CC Example

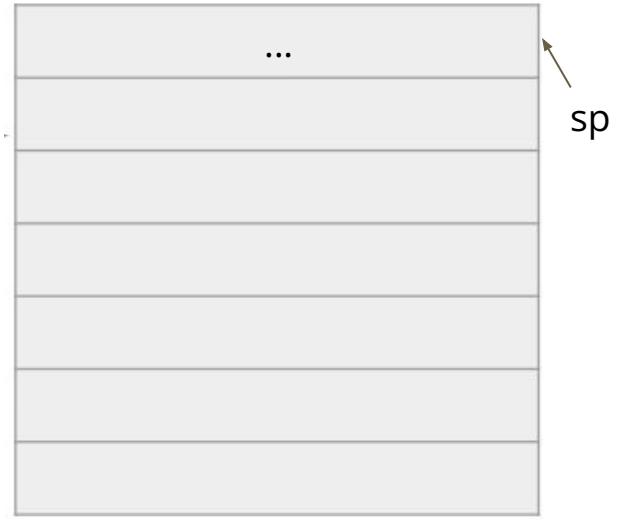
```
func1: # modifies a0, t0, and s0. ra points to the `main` function.
    # Checkpoint 1: Prologue omitted
    # Checkpoint 2: Saving t0 omitted
    # Checkpoint 3: Restoring t0 omitted
    # Checkpoint 4: Epilogue
    jr ra   # function return
```



What should we do for the epilogue

CC Example

```
func1: # modifies a0, t0, and s0. ra points to the `main` function.
    # Checkpoint 1: Prologue omitted
    # Checkpoint 2: Saving t0 omitted
    # Checkpoint 3: Restoring t0 omitted
    # Checkpoint 4: Epilogue
    lw s0 4(sp)     # Retrieve the original saved register (s0)
    lw ra 0(sp)     # Retrieve the original return address (ra)
    addi sp sp 8    # Return the stack pointer up by 2 words (8 bytes)


    jr ra   # function return
```



CC Example

```
func1: # modifies a0, t0, and s0. ra points to the `main` function.
    # Checkpoint 1: What do you need to do before you start modifying registers?
    addi sp sp -8   # Push the stack pointer down by 2 words (8 bytes)
    sw ra 0(sp)     # Save the return address register (ra)
    sw s0 4(sp)     # Save the saved register (s0)

    # Some block of code using a0, t0, and s0
    # Checkpoint 2: What do you need to do before you call another function?
    addi sp sp -4   # Push the stack pointer down by 1 word (4 bytes)
    sw t0 0(sp)     # Save the temporary register (t0)

    # input argument at a0, return value at a0
    jal ra, func2  # call func2
    # Checkpoint 3: What do you need to do after a function call?
    lw t0 0(sp)     # Retrieve the saved temporary register from the stack
    addi sp sp 4    # Return the stack pointer up by 1 word (4 bytes)

    # Some block of code using a0, t0, and s0
    # Checkpoint 4: What do you need to do before this function returns?
    lw s0 4(sp)     # Retrieve the original saved register (s0)
    lw ra 0(sp)     # Retrieve the original return address (ra). This points back to the main function.
    addi sp sp 8    # Return the stack pointer up by 2 words (8 bytes)

    jr ra   # function return
```

Another Example C->RISC-V

```c
int source[] = {3, 1, 4, 1, 5, 9, 0};
int dest[10];

int fun(int x) {
        return -x * (x + 1);
}

int main() {
    int k;
    int sum = 0;
    for (k = 0; source[k] != 0; k++) {
        dest[k] = fun(source[k]);
        sum += dest[k];
    }
    printf("sum: %d\n", sum);
}
```

Lets convert this to RISC-V

Another Example C->RISC-V

```c
int source[] = {3, 1, 4, 1, 5, 9, 0};
int dest[10];
```



```
.data
source:
    .word   3
    .word   1
    .word   4
    .word   1
    .word   5
    .word   9
    .word   0
dest:
    .word   0
    .word   0
    .word   0
    .word   0
    .word   0
    .word   0
    .word   0
    .word   0
    .word   0
    .word   0
```

We create source and dest as global vars under .data

Another Example C->RISC-V

```
int fun(int x) {
        return -x * (x + 1);
}
```


Lets translate the fun function
a0 contains x
a0 will contain return val

Another Example C->RISC-V

```c
int fun(int x) {
        return -x * (x + 1);
}
```



```
.text
fun:
    addi t0, a0, 1 # t0 = x + 1
    sub t1, x0, a0 # t1 = -x
    mul a0, t0, t1 # a0 = (x + 1) * (-x)
    jr ra # return
```

Lets translate the fun function
a0 contains x
a0 will contain return val

Another Example C->RISC-V

```c
int main() {
    int k;
    int sum = 0;
    for (k = 0; source[k] != 0; k++) {
        dest[k] = fun(source[k]);
        sum += dest[k];
    }
```



```
main:
    addi t0, x0, 0 # t0 = k = 0
    addi s0, x0, 0 # s0 = sum = 0
    la s1, source
    la s2, dest
loop:
#1  slli s3, t0, 2
#2  add t1, s1, s3
#3  lw t2, 0(t1)
#4  beq t2, x0, exit
    ...
#5  addi t0, t0, 1
#6  jal x0, loop
exit:
```

We ignore Calling Conventions for now
Outer body of loop:

Another Example C->RISC-V

```c
int main() {
    int k;
    int sum = 0;
    for (k = 0; source[k] != 0; k++) {
        dest[k] = fun(source[k]);
        sum += dest[k];
    }
```



```
main:
    addi t0, x0, 0 # t0 = k = 0
    addi s0, x0, 0 # s0 = sum = 0
    la s1, source
    la s2, dest
loop:
    slli s3, t0, 2
    add t1, s1, s3
    lw t2, 0(t1)
    beq t2, x0, exit
#1  add a0, x0, t2 # 1
    ...
#2  jal fun # 2
    ...
#3  add t3, s2, s3 # 4
#4  sw a0, 0(t3) # 5
#5  add s0, s0, a0 # 6
    addi t0, t0, 1
    jal x0, loop
exit:
```

Filling in the rest of the loop, still ignoring calling conventions

Another Example C->RISC-V

```c
int main() {
    int k;
    int sum = 0;
    for (k = 0; source[k] != 0; k++) {
        dest[k] = fun(source[k]);
        sum += dest[k];
    }
```



```
loop:
    slli s3, t0, 2
    add t1, s1, s3
    lw t2, 0(t1)
    beq t2, x0, exit
#1  add a0, x0, t2 # 1
    addi sp, sp, -4
    sw t0, 0(sp)
    jal fun
    lw t0, 0(sp)
    addi sp, sp, 4
#3  add t3, s2, s3 # 4
#4  sw a0, 0(t3) # 5
#5  add s0, s0, a0 # 6
    addi t0, t0, 1
    jal x0, loop
exit:
    addi a0, x0, 1 # argument to ecall, 1 = execute print integer
    addi a1, s0, 0 # argument to ecall, the value to be printed
    ecall # print integer ecall
    addi a0, x0, 10 # argument to ecall, 10 = terminate program
    ecall # terminate program
```

We then move onto exit without considering calling conventions first

Another Example C->RISC-V

```c
int main() {
    int k;
    int sum = 0;
    for (k = 0; source[k] != 0; k++) {
        dest[k] = fun(source[k]);
        sum += dest[k];
    }
```



```
main:
    # BEGIN PROLOGUE
    addi sp, sp, -20
    sw s0, 0(sp)
    sw s1, 4(sp)
    sw s2, 8(sp)
    sw s3, 12(sp)
    sw ra, 16(sp)
    # END PROLOGUE
    ...
    ...
exit:
    addi a0, x0, 1 # argument to ecall, 1 = execute print integer
    addi a1, s0, 0 # argument to ecall, the value to be printed
    ecall # print integer ecall
    # BEGIN EPILOGUE
    lw s0, 0(sp)
    lw s1, 4(sp)
    lw s2, 8(sp)
    lw s3, 12(sp)
    lw ra, 16(sp)
    addi sp, sp, 20
    # END EPILOGUE
    addi a0, x0, 10 # argument to ecall, 10 = terminate program
    ecall # terminate program
```

Finally, we fill out prologue and epilogue