ECE 550D
Fundamentals of Computer Systems and Engineering

Fall 2023

Instruction Set Architectures (ISAs) and MIPS

Xin Li & Dawei Liu
Duke Kunshan University

Slides are derived from work by
Andrew Hilton, Tyler Bletsch and Rabih Younes (Duke)

Last time…

- Who can remind us what we did last time?
  - MIPS ISA and Assembly Programming
  - More on Assembly Programming today

Execution example: Calling with frames

Addr	Instruction
1000	subiu $sp, $sp, 16
1004	sw $fp, 0($sp)
1008	sw $ra, 4($sp)
100C	sw $s0, 8($sp)
1010	addiu $fp, $sp, 12
1014	add $s0, $a0, $a1
1018	jal 4200
101C	add $t0, $v0, $s0
1020	lw $v0, 4($t0)
1024	lw $s0, -4($fp)
1028	lw $ra, -8($fp)
102C	lw $fp, -12($fp)
1030	addiu $sp, $sp, 16
1034	jr $ra

Just did jal 1000

Reg	Value
$0	0000 0000
$at	0000 0000
$v0	4242 4242
$v1	0000 8010
$a0	0000 1234
$a1	5678 0001
$a2	0000 0002
$a3	0000 0007
$t0	9999 999A
$t1	0000 0000
$s0	0042 0420
$sp	0000 FFE0
$fp	0000 FFF0
$ra	0000 2348
PC	0000 1000

Addr	Value
FFF0	0001 0070
FFEC	1234 5678
FFE8	9999 9999
FFE4	0000 2568
FFE0	0001 0040
FFDC	
FFD8	
FFD4	
FFD0	
FFCC	
FFC8	
FFC4	
FFC0	
FFBC	

Execution example: Calling with frames

Addr	Instruction
1000	subiu $sp, $sp, 16
1004	sw $fp, 0($sp)
1008	sw $ra, 4($sp)
100C	sw $s0, 8($sp)
1010	addiu $fp, $sp, 12
1014	add $s0, $a0, $a1
1018	jal 4200
101C	add $t0, $v0, $s0
1020	lw $v0, 4($t0)
1024	lw $s0, -4($fp)
1028	lw $ra, -8($fp)
102C	lw $fp, -12($fp)
1030	addiu $sp, $sp, 16
1034	jr $ra

$sp, $fp still describe callers frame

Reg	Value
$0	0000 0000
$at	0000 0000
$v0	4242 4242
$v1	0000 8010
$a0	0000 1234
$a1	5678 0001
$a2	0000 0002
$a3	0000 0007
$t0	9999 999A
$t1	0000 0000
$s0	0042 0420
$sp	0000 FFE0
$fp	0000 FFF0
$ra	0000 2348
PC	0000 1000

Addr	Value
FFF0	0001 0070
FFEC	1234 5678
FFE8	9999 9999
FFE4	0000 2568
FFE0	0001 0040
FFDC	
FFD8	
FFD4	
FFD0	
FFCC	
FFC8	
FFC4	
FFC0	
FFBC	

Execution example: Calling with frames

Addr	Instruction
1000	subiu $sp, $sp, 16
1004	sw $fp, 0($sp)
1008	sw $ra, 4($sp)
100C	sw $s0, 8($sp)
1010	addiu $fp, $sp, 12
1014	add $s0, $a0, $a1
1018	jal 4200
101C	add $t0, $v0, $s0
1020	lw $v0, 4($t0)
1024	lw $s0, -4($fp)
1028	lw $ra, -8($fp)
102C	lw $fp, -12($fp)
1030	addiu $sp, $sp, 16
1034	jr $ra

Allocate space on stack

Reg	Value
$0	0000 0000
$at	0000 0000
$v0	4242 4242
$v1	0000 8010
$a0	0000 1234
$a1	5678 0001
$a2	0000 0002
$a3	0000 0007
$t0	9999 999A
$t1	0000 0000
$s0	0042 0420
$sp	**0000 FFD0**
$fp	0000 FFF0
$ra	0000 2348
PC	0000 1004

Addr	Value
FFF0	0001 0070
FFEC	1234 5678
FFE8	9999 9999
FFE4	0000 2568
FFE0	0001 0040
FFDC	
FFD8	
FFD4	
FFD0	
FFCC	
FFC8	
FFC4	
FFC0	
FFBC	

Execution example: Calling with frames

Addr	Instruction
1000	subiu $sp, $sp, 16
1004	sw $fp, 0($sp)
1008	sw $ra, 4($sp)
100C	sw $s0, 8($sp)
1010	addiu $fp, $sp, 12
1014	add $s0, $a0, $a1
1018	jal 4200
101C	add $t0, $v0, $s0
1020	lw $v0, 4($t0)
1024	lw $s0, -4($fp)
1028	lw $ra, -8($fp)
102C	lw $fp, -12($fp)
1030	addiu $sp, $sp, 16
1034	jr $ra

Save $fp

Reg	Value
$0	0000 0000
$at	0000 0000
$v0	4242 4242
$v1	0000 8010
$a0	0000 1234
$a1	5678 0001
$a2	0000 0002
$a3	0000 0007
$t0	9999 999A
$t1	0000 0000
$s0	0042 0420
$sp	0000 FFD0
$fp	0000 FFF0
$ra	0000 2348
PC	0000 1008

Addr	Value
FFF0	0001 0070
FFEC	1234 5678
FFE8	9999 9999
FFE4	0000 2568
FFE0	0001 0040
FFDC	
FFD8	
FFD4	
FFD0	**0000 FFF0**
FFCC	
FFC8	
FFC4	
FFC0	
FFBC	

Execution example: Calling with frames

Addr	Instruction
1000	subiu $sp, $sp, 16
1004	sw $fp, 0($sp)
1008	sw $ra, 4($sp)
100C	sw $s0, 8($sp)
1010	addiu $fp, $sp, 12
1014	add $s0, $a0, $a1
1018	jal 4200
101C	add $t0, $v0, $s0
1020	lw $v0, 4($t0)
1024	lw $s0, -4($fp)
1028	lw $ra, -8($fp)
102C	lw $fp, -12($fp)
1030	addiu $sp, $sp, 16
1034	jr $ra

Save $ra

Reg	Value
$0	0000 0000
$at	0000 0000
$v0	4242 4242
$v1	0000 8010
$a0	0000 1234
$a1	5678 0001
$a2	0000 0002
$a3	0000 0007
$t0	9999 999A
$t1	0000 0000
$s0	0042 0420
$sp	0000 FFD0
$fp	0000 FFF0
$ra	0000 2348
PC	0000 100C

Addr	Value
FFF0	0001 0070
FFEC	1234 5678
FFE8	9999 9999
FFE4	0000 2568
FFE0	0001 0040
FFDC	
FFD8	
FFD4	**0000 2348**
FFD0	0000 FFF0
FFCC	
FFC8	
FFC4	
FFC0	
FFBC	

Execution example: Calling with frames

Addr	Instruction
1000	subiu $sp, $sp, 16
1004	sw $fp, 0($sp)
1008	sw $ra, 4($sp)
100C	sw $s0, 8($sp)
1010	addiu $fp, $sp, 12
1014	add $s0, $a0, $a1
1018	jal 4200
101C	add $t0, $v0, $s0
1020	lw $v0, 4($t0)
1024	lw $s0, -4($fp)
1028	lw $ra, -8($fp)
102C	lw $fp, -12($fp)
1030	addiu $sp, $sp, 16
1034	jr $ra

Save $s0 (callee saves)

Reg	Value
$0	0000 0000
$at	0000 0000
$v0	4242 4242
$v1	0000 8010
$a0	0000 1234
$a1	5678 0001
$a2	0000 0002
$a3	0000 0007
$t0	9999 999A
$t1	0000 0000
$s0	0042 0420
$sp	0000 FFD0
$fp	0000 FFF0
$ra	0000 2348
PC	0000 1010

Addr	Value
FFF0	0001 0070
FFEC	1234 5678
FFE8	9999 9999
FFE4	0000 2568
FFE0	0001 0040
FFDC	
FFD8	**0042 0420**
FFD4	0000 2348
FFD0	0000 FFF0
FFCC	
FFC8	
FFC4	
FFC0	
FFBC	

Execution example: Calling with frames

Addr	Instruction
1000	subiu $sp, $sp, 16
1004	sw $fp, 0($sp)
1008	sw $ra, 4($sp)
100C	sw $s0, 8($sp)
1010	addiu $fp, $sp, 12
1014	add $s0, $a0, $a1
1018	jal 4200
101C	add $t0, $v0, $s0
1020	lw $v0, 4($t0)
1024	lw $s0, -4($fp)
1028	lw $ra, -8($fp)
102C	lw $fp, -12($fp)
1030	addiu $sp, $sp, 16
1034	jr $ra

Setup $fp

Reg	Value
$0	0000 0000
$at	0000 0000
$v0	4242 4242
$v1	0000 8010
$a0	0000 1234
$a1	5678 0001
$a2	0000 0002
$a3	0000 0007
$t0	9999 999A
$t1	0000 0000
$s0	0042 0420
$sp	0000 FFD0
$fp	**0000 FFDC**
$ra	0000 2348
PC	0000 1014

Addr	Value
FFF0	0001 0070
FFEC	1234 5678
FFE8	9999 9999
FFE4	0000 2568
FFE0	0001 0040
FFDC	
FFD8	0042 0420
FFD4	0000 2348
FFD0	0000 FFF0
FFCC	
FFC8	
FFC4	
FFC0	
FFBC	

Execution example: Calling with frames

Addr	Instruction
1000	subiu $sp, $sp, 16
1004	sw $fp, 0($sp)
1008	sw $ra, 4($sp)
100C	sw $s0, 8($sp)
1010	addiu $fp, $sp, 12
1014	add $s0, $a0, $a1
1018	jal 4200
101C	add $t0, $v0, $s0
1020	lw $v0, 4($t0)
1024	lw $s0, -4($fp)
1028	lw $ra, -8($fp)
102C	lw $fp, -12($fp)
1030	addiu $sp, $sp, 16
1034	jr $ra

$sp, $fp now describe new frame, ready to start

Reg	Value
$0	0000 0000
$at	0000 0000
$v0	4242 4242
$v1	0000 8010
$a0	0000 1234
$a1	5678 0001
$a2	0000 0002
$a3	0000 0007
$t0	9999 999A
$t1	0000 0000
$s0	0042 0420
$sp	0000 FFD0
$fp	0000 FFDC
$ra	0000 2348
PC	0000 1014

Addr	Value
FFF0	0001 0070
FFEC	1234 5678
FFE8	9999 9999
FFE4	0000 2568
FFE0	0001 0040
FFDC	
FFD8	0042 0420
FFD4	0000 2348
FFD0	0000 FFF0
FFCC	
FFC8	
FFC4	
FFC0	
FFBC	

Execution example: Calling with frames

Addr	Instruction
1000	subiu $sp, $sp, 16
1004	sw $fp, 0($sp)
1008	sw $ra, 4($sp)
100C	sw $s0, 8($sp)
1010	addiu $fp, $sp, 12
1014	add $s0, $a0, $a1
1018	jal 4200
101C	add $t0, $v0, $s0
1020	lw $v0, 4($t0)
1024	lw $s0, -4($fp)
1028	lw $ra, -8($fp)
102C	lw $fp, -12($fp)
1030	addiu $sp, $sp, 16
1034	jr $ra

Do some computation..

Reg	Value
$0	0000 0000
$at	0000 0000
$v0	4242 4242
$v1	0000 8010
$a0	0000 1234
$a1	5678 0001
$a2	0000 0002
$a3	0000 0007
$t0	9999 999A
$t1	0000 0000
$s0	**5678 1235**
$sp	0000 FFD0
$fp	0000 FFDC
$ra	0000 2348
PC	0000 1018

Addr	Value
FFF0	0001 0070
FFEC	1234 5678
FFE8	9999 9999
FFE4	0000 2568
FFE0	0001 0040
FFDC	
FFD8	0042 0420
FFD4	0000 2348
FFD0	0000 FFF0
FFCC	
FFC8	
FFC4	
FFC0	
FFBC	

Execution example: Calling with frames

Addr	Instruction
1000	subiu $sp, $sp, 16
1004	sw $fp, 0($sp)
1008	sw $ra, 4($sp)
100C	sw $s0, 8($sp)
1010	addiu $fp, $sp, 12
1014	add $s0, $a0, $a1
1018	jal 4200
101C	add $t0, $v0, $s0
1020	lw $v0, 4($t0)
1024	lw $s0, -4($fp)
1028	lw $ra, -8($fp)
102C	lw $fp, -12($fp)
1030	addiu $sp, $sp, 16
1034	jr $ra

Call another function
(not pictured, takes no args)

Reg	Value
$0	0000 0000
$at	0000 0000
$v0	4242 4242
$v1	0000 8010
$a0	0000 1234
$a1	5678 0001
$a2	0000 0002
$a3	0000 0007
$t0	9999 999A
$t1	0000 0000
$s0	5678 1235
$sp	0000 FFD0
$fp	0000 FFDC
$ra	**0000 101C**
PC	**0000 4200**

jal sets $ra, PC

Addr	Value
FFF0	0001 0070
FFEC	1234 5678
FFE8	9999 9999
FFE4	0000 2568
FFE0	0001 0040
FFDC	
FFD8	0042 0420
FFD4	0000 2348
FFD0	0000 FFF0
FFCC	
FFC8	
FFC4	
FFC0	
FFBC	

Execution example: Calling with frames

Addr	Instruction
1000	subiu $sp, $sp, 16
1004	sw $fp, 0($sp)
1008	sw $ra, 4($sp)
100C	sw $s0, 8($sp)
1010	addiu $fp, $sp, 12
1014	add $s0, $a0, $a1
1018	jal 4200
101C	add $t0, $v0, $s0
1020	lw $v0, 4($t0)
1024	lw $s0, -4($fp)
1028	lw $ra, -8($fp)
102C	lw $fp, -12($fp)
1030	addiu $sp, $sp, 16
1034	jr $ra

Other function can do what
It wants to the regs as it computes

Reg	Value
$0	0000 0000
$at	???? ????
$v0	???? ????
$v1	???? ????
$a0	???? ????
$a1	???? ????
$a2	???? ????
$a3	???? ????
$t0	???? ????
$t1	???? ????
$s0	???? ????
$sp	???? ????
$fp	???? ????
$ra	???? ????
PC	???? ????

Addr	Value
FFF0	0001 0070
FFEC	1234 5678
FFE8	9999 9999
FFE4	0000 2568
FFE0	0001 0040
FFDC	
FFD8	0042 0420
FFD4	0000 2348
FFD0	0000 FFF0
FFCC	
FFC8	
FFC4	
FFC0	
FFBC	

And make a stack frame
Of its own

Execution example: Calling with frames

Addr	Instruction
1000	subiu $sp, $sp, 16
1004	sw $fp, 0($sp)
1008	sw $ra, 4($sp)
100C	sw $s0, 8($sp)
1010	addiu $fp, $sp, 12
1014	add $s0, $a0, $a1
1018	jal 4200
101C	add $t0, $v0, $s0
1020	lw $v0, 4($t0)
1024	lw $s0, -4($fp)
1028	lw $ra, -8($fp)
102C	lw $fp, -12($fp)
1030	addiu $sp, $sp, 16
1034	jr $ra

But before it returns, it is
responsible for restoring certain
registers

Reg	Value
$0	0000 0000
$at	???? ????
$v0	**8675 3090**
$v1	???? ????
$a0	???? ????
$a1	???? ????
$a2	???? ????
$a3	???? ????
$t0	???? ????
$t1	???? ????
$s0	**5678 1235**
$sp	**0000 FFD0**
$fp	**0000 FFDC**
$ra	**0000 101C**
PC	**0000 101C**

Addr	Value
FFF0	0001 0070
FFEC	1234 5678
FFE8	9999 9999
FFE4	0000 2568
FFE0	0001 0040
FFDC	
FFD8	0042 0420
FFD4	0000 2348
FFD0	0000 FFF0
FFCC	
FFC8	
FFC4	
FFC0	
FFBC	

Including $sp and $fp,
and $s0
Value returned in $v0

Execution example: Calling with frames

Addr	Instruction
1000	subiu $sp, $sp, 16
1004	sw $fp, 0($sp)
1008	sw $ra, 4($sp)
100C	sw $s0, 8($sp)
1010	addiu $fp, $sp, 12
1014	add $s0, $a0, $a1
1018	jal 4200
101C	add $t0, $v0, $s0
1020	lw $v0, 4($t0)
1024	lw $s0, -4($fp)
1028	lw $ra, -8($fp)
102C	lw $fp, -12($fp)
1030	addiu $sp, $sp, 16
1034	jr $ra

Do some more computation

Reg	Value
$0	0000 0000
$at	???? ????
$v0	8675 3090
$v1	???? ????
$a0	???? ????
$a1	???? ????
$a2	???? ????
$a3	???? ????
$t0	**DCED 42C5**
$t1	???? ????
$s0	5678 1235
$sp	0000 FFD0
$fp	0000 FFDC
$ra	0000 101C
PC	0000 1020

Addr	Value
FFF0	0001 0070
FFEC	1234 5678
FFE8	9999 9999
FFE4	0000 2568
FFE0	0001 0040
FFDC	
FFD8	0042 0420
FFD4	0000 2348
FFD0	0000 FFF0
FFCC	
FFC8	
FFC4	
FFC0	
FFBC	

Execution example: Calling with frames

Addr	Instruction
1000	subiu $sp, $sp, 16
1004	sw $fp, 0($sp)
1008	sw $ra, 4($sp)
100C	sw $s0, 8($sp)
1010	addiu $fp, $sp, 12
1014	add $s0, $a0, $a1
1018	jal 4200
101C	add $t0, $v0, $s0
1020	lw $v0, 4($t0)
1024	lw $s0, -4($fp)
1028	lw $ra, -8($fp)
102C	lw $fp, -12($fp)
1030	addiu $sp, $sp, 16
1034	jr $ra

Do some more computation
(load addr not pictured)

Reg	Value
$0	0000 0000
$at	???? ????
$v0	**0001 0002**
$v1	???? ????
$a0	???? ????
$a1	???? ????
$a2	???? ????
$a3	???? ????
$t0	DCED 42C5
$t1	???? ????
$s0	5678 1235
$sp	0000 FFD0
$fp	0000 FFDC
$ra	0000 101C
PC	0000 1024

Addr	Value
FFF0	0001 0070
FFEC	1234 5678
FFE8	9999 9999
FFE4	0000 2568
FFE0	0001 0040
FFDC	
FFD8	0042 0420
FFD4	0000 2348
FFD0	0000 FFF0
FFCC	
FFC8	
FFC4	
FFC0	
FFBC	

Execution example: Calling with frames

Addr	Instruction
1000	subiu $sp, $sp, 16
1004	sw $fp, 0($sp)
1008	sw $ra, 4($sp)
100C	sw $s0, 8($sp)
1010	addiu $fp, $sp, 12
1014	add $s0, $a0, $a1
1018	jal 4200
101C	add $t0, $v0, $s0
1020	lw $v0, 4($t0)
1024	lw $s0, -4($fp)
1028	lw $ra, -8($fp)
102C	lw $fp, -12($fp)
1030	addiu $sp, $sp, 16
1034	jr $ra

Restore registers to return

Reg	Value
$0	0000 0000
$at	???? ????
$v0	0001 0002
$v1	???? ????
$a0	???? ????
$a1	???? ????
$a2	???? ????
$a3	???? ????
$t0	DCED 42C5
$t1	???? ????
$s0	**0042 0420**
$sp	0000 FFD0
$fp	0000 FFDC
$ra	0000 101C
PC	0000 1028

Addr	Value
FFF0	0001 0070
FFEC	1234 5678
FFE8	9999 9999
FFE4	0000 2568
FFE0	0001 0040
FFDC	
FFD8	0042 0420
FFD4	0000 2348
FFD0	0000 FFF0
FFCC	
FFC8	
FFC4	
FFC0	
FFBC	

Execution example: Calling with frames

Addr	Instruction
1000	subiu $sp, $sp, 16
1004	sw $fp, 0($sp)
1008	sw $ra, 4($sp)
100C	sw $s0, 8($sp)
1010	addiu $fp, $sp, 12
1014	add $s0, $a0, $a1
1018	jal 4200
101C	add $t0, $v0, $s0
1020	lw $v0, 4($t0)
1024	lw $s0, -4($fp)
1028	lw $ra, -8($fp)
102C	lw $fp, -12($fp)
1030	addiu $sp, $sp, 16
1034	jr $ra

Restore registers to return

Reg	Value
$0	0000 0000
$at	???? ????
$v0	0001 0002
$v1	???? ????
$a0	???? ????
$a1	???? ????
$a2	???? ????
$a3	???? ????
$t0	DCED 42C5
$t1	???? ????
$s0	0042 0420
$sp	0000 FFD0
$fp	0000 FFDC
$ra	**0000 2348**
PC	0000 102C

Addr	Value
FFF0	0001 0070
FFEC	1234 5678
FFE8	9999 9999
FFE4	0000 2568
FFE0	0001 0040
FFDC	
FFD8	0042 0420
FFD4	0000 2348
FFD0	0000 FFF0
FFCC	
FFC8	
FFC4	
FFC0	
FFBC	

Execution example: Calling with frames

Addr	Instruction
1000	subiu $sp, $sp, 16
1004	sw $fp, 0($sp)
1008	sw $ra, 4($sp)
100C	sw $s0, 8($sp)
1010	addiu $fp, $sp, 12
1014	add $s0, $a0, $a1
1018	jal 4200
101C	add $t0, $v0, $s0
1020	lw $v0, 4($t0)
1024	lw $s0, -4($fp)
1028	lw $ra, -8($fp)
102C	lw $fp, -12($fp)
1030	addiu $sp, $sp, 16
1034	jr $ra

Restore registers to return

Reg	Value
$0	0000 0000
$at	???? ????
$v0	0001 0002
$v1	???? ????
$a0	???? ????
$a1	???? ????
$a2	???? ????
$a3	???? ????
$t0	DCED 42C5
$t1	???? ????
$s0	0042 0420
$sp	0000 FFD0
$fp	**0000 FFF0**
$ra	0000 2348
PC	0000 1030

Addr	Value
FFF0	0001 0070
FFEC	1234 5678
FFE8	9999 9999
FFE4	0000 2568
FFE0	0001 0040
FFDC	
FFD8	0042 0420
FFD4	0000 2348
FFD0	0000 FFF0
FFCC	
FFC8	
FFC4	
FFC0	
FFBC	

Execution example: Calling with frames

Addr	Instruction
1000	subiu $sp, $sp, 16
1004	sw $fp, 0($sp)
1008	sw $ra, 4($sp)
100C	sw $s0, 8($sp)
1010	addiu $fp, $sp, 12
1014	add $s0, $a0, $a1
1018	jal 4200
101C	add $t0, $v0, $s0
1020	lw $v0, 4($t0)
1024	lw $s0, -4($fp)
1028	lw $ra, -8($fp)
102C	lw $fp, -12($fp)
1030	addiu $sp, $sp, 16
1034	jr $ra

Restore registers to return

Reg	Value
$0	0000 0000
$at	???? ????
$v0	0001 0002
$v1	???? ????
$a0	???? ????
$a1	???? ????
$a2	???? ????
$a3	???? ????
$t0	DCED 42C5
$t1	???? ????
$s0	0042 0420
$sp	**0000 FFE0**
$fp	0000 FFF0
$ra	0000 2348
PC	0000 1034

Addr	Value
FFF0	0001 0070
FFEC	1234 5678
FFE8	9999 9999
FFE4	0000 2568
FFE0	0001 0040
FFDC	
FFD8	0042 0420
FFD4	0000 2348
FFD0	0000 FFF0
FFCC	
FFC8	
FFC4	
FFC0	
FFBC	

Execution example: Calling with frames

Addr	Instruction
1000	subiu $sp, $sp, 16
1004	sw $fp, 0($sp)
1008	sw $ra, 4($sp)
100C	sw $s0, 8($sp)
1010	addiu $fp, $sp, 12
1014	add $s0, $a0, $a1
1018	jal 4200
101C	add $t0, $v0, $s0
1020	lw $v0, 4($t0)
1024	lw $s0, -4($fp)
1028	lw $ra, -8($fp)
102C	lw $fp, -12($fp)
1030	addiu $sp, $sp, 16
1034	jr $ra

Restore registers to return

Reg	Value
$0	0000 0000
$at	???? ????
$v0	0001 0002
$v1	???? ????
$a0	???? ????
$a1	???? ????
$a2	???? ????
$a3	???? ????
$t0	DCED 42C5
$t1	???? ????
$s0	0042 0420
$sp	0000 FFE0
$fp	0000 FFF0
$ra	0000 2348
PC	0000 1034

Addr	Value
FFF0	0001 0070
FFEC	1234 5678
FFE8	9999 9999
FFE4	0000 2568
FFE0	0001 0040
FFDC	
FFD8	0042 0420
FFD4	0000 2348
FFD0	0000 FFF0
FFCC	
FFC8	
FFC4	
FFC0	
FFBC	

Now $sp, $fp describe caller's frame

Execution example: Calling with frames

Addr	Instruction
1000	subiu $sp, $sp, 16
1004	sw $fp, 0($sp)
1008	sw $ra, 4($sp)
100C	sw $s0, 8($sp)
1010	addiu $fp, $sp, 12
1014	add $s0, $a0, $a1
1018	jal 4200
101C	add $t0, $v0, $s0
1020	lw $v0, 4($t0)
1024	lw $s0, -4($fp)
1028	lw $ra, -8($fp)
102C	lw $fp, -12($fp)
1030	addiu $sp, $sp, 16
1034	jr $ra

Return to caller
(code not pictured)

Reg	Value
$0	0000 0000
$at	???? ????
$v0	0001 0002
$v1	???? ????
$a0	???? ????
$a1	???? ????
$a2	???? ????
$a3	???? ????
$t0	DCED 42C5
$t1	???? ????
$s0	0042 0420
$sp	0000 FFE0
$fp	0000 FFF0
$ra	0000 2348
PC	0000 2348

Addr	Value
FFF0	0001 0070
FFEC	1234 5678
FFE8	9999 9999
FFE4	0000 2568
FFE0	0001 0040
FFDC	
FFD8	0042 0420
FFD4	0000 2348
FFD0	0000 FFF0
FFCC	
FFC8	
FFC4	
FFC0	
FFBC	

Assembly Writing Tips and Advice

- Write C first, translate C -> Assembly
  - One function at a time
  - Pick registers for each variable
    - Must be in memory? Give it a stack slot (refer to by $fp+num)
  - Write prolog
    - Save ra/fp (if needed)
    - Save any $s registers you use
  - Translate line by line
  - Write epilog

Why do we need FP?

- The frame pointer is not always required
  - Can often get away without it
- When/why do we need it?
  - Debugging tools (gdb) use it to find frames
  - If you have variable length arrays
    - Stack pointer changes by amount not know at compile time
    - Variables still at constant offset from frame pointer
- How to reference stuff without it?
  - Everything is offset from the stack pointer: -4($sp), -8($sp), etc.
- Good practice for this class to use it
  - Don't prematurely optimize

System Call Instruction

- System call is used to communicate with the operating system and request services (memory allocation, I/O)
  - syscall instruction in MIPS
- Sort of like a procedure call, but call to ask OS for help
- SPIM supports "system calls lite"
  1. Load system call code into register $v0
     - Example: if $v0==1, then syscall will print an integer
  2. Load arguments (if any) into registers $a0, $a1, or $f12 (for floating point)
  3. `syscall`
  4. Results returned in registers $v0 or $f0

SPIM System Call Support

code	service	ArgType	Arg/Result
1	print	int	$a0
2	print	float	$f12
3	print	double	$f12
4	print	string	$a0 (string address)
5	read	integer	integer in $v0
6	read	float	float in $f0
7	read	double	double in $f0
8	read	string	$a0=buffer, $a1=length
9	sbrk	$a0=amount	address in $v0
10	exit		

Plus a few more for general file IO which we shouldn't need.

Echo number and string

```
.text
main:
    li      $v0, 5              # code to read an integer
    syscall             # do the read (invokes the OS)
    move     $a0, $v0   # copy result from $v0 to $a0

    li      $v0, 1              # code to print an integer
    syscall             # print the integer

    li      $v0, 4              # code to print string
    la      $a0, nln    # address of string (newline)
    syscall

    li      $v0, 8              # code to read a string
    la      $a0, name   # address of buffer (name)
    li      $a1, 8              # size of buffer (8 bytes)
    syscall

    la      $a0, name   # address of string to print
    li      $v0, 4              # code to print a string
    syscall

    jr      $31                 # return

    .data
    .align 2
name:       .word 0,0
nln:        .asciiz "\n"
```


MIPS Assembly General Rules

- One instruction per line.
- Numbers are base-10 integers or Hex w/ leading 0x.
- Identifiers: alphanumeric, _, . string starting in a letter or _
- Labels: identifiers starting at the beginning of a line followed by ":"
- Comments: everything following # till end-of-line.
- Instruction format: Space and "," separated fields.
  - [Label:] <op> reg1, [reg2], [reg3] [# comment]
  - [Label:] <op> reg1, offset(reg2) [# comment]
  - .Directive [arg1], [arg2], . . .

Summary

- MIPS ISA and Assembly Programming
  - We'll use qtspim (with spim for automated testing)
  - Have seen most basic instruction types

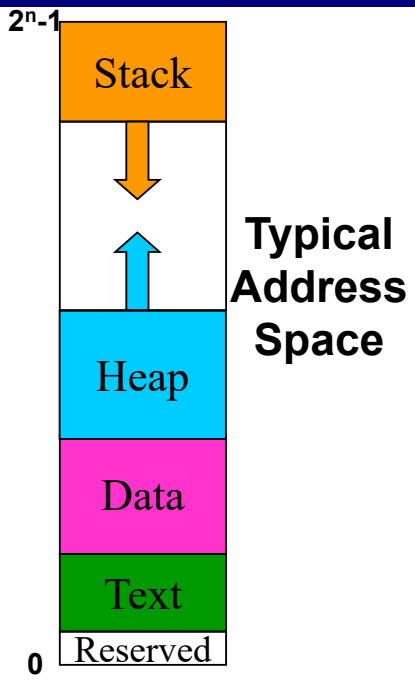
- Example MIPS programs:
  - simple.s:
    http://people.duke.edu/~tkb13/courses/ece550-2016fa/resources/simple.s
  - sum = sum + i*i:
    https://www.cs.duke.edu/courses/fall13/compsci250/code/sum_i_sqr_slt.s
  - sum array:
    https://www.cs.duke.edu/courses/fall13/compsci250/code/sum_array_slt.s
  - recursive sum array:
    https://www.cs.duke.edu/courses/fall13/compsci250/code/recur_sum_array.s
  - recursive sum i*i:
    https://www.cs.duke.edu/courses/fall13/compsci250/code/recur_sum_i_sqr.s

Mapping C variables to memory

Supplementary Materials

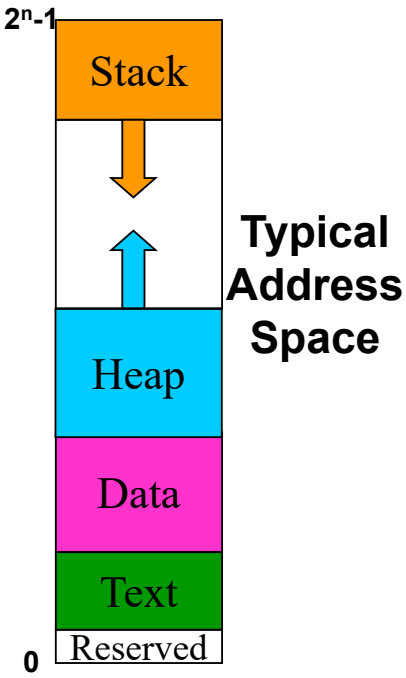
Memory Layout

- Memory is array of bytes, but there are conventions as to what goes where in this array
- Text: instructions (the program to execute)
- Data: global variables
- Stack: local variables and other per-function state; starts at top & grows down
- Heap: dynamically allocated variables; grows up
- What if stack and heap overlap????



Memory Layout: Example

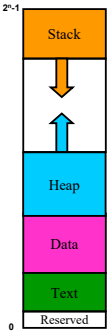
```
int anumber = 3;

int factorial (int x) {
  if (x == 0) {
    return 1;
  }
  else {
    return x * factorial (x - 1);
  }
}

int main (void) {
  int z = factorial (anumber);
  printf("%d\n", z);
  return 0;
}
```



Memory Layout: Example

```
Global  int anumber = 3;

Param   int factorial (int x) {
          if (x == 0) {
            return 1;
          }
          else {
            return x * factorial (x - 1);
          }
        }

        int main (void) {
Local     int z = factorial (anumber);
          printf("%d\n", z);
          return 0;
        }
```

```
.data
anumber: .word 3

.text
.globl factorial
factorial:
; find x in $a0.
; if more args than 4,
; find them in stack
  ...

.globl main
main:
; make local z exist by
; subtracting 4 from $sp
addiu $sp, $sp, -4
  ...
```



What is array?

- Array is nothing but a contiguous chunk of memory.
- The name of the array is a pointer to the beginning of the chunk of memory array has.
- So int array[100] is a contiguous chunk of memory which is of size 100*4 = 400.
- so array is almost of type int*
  (not exactly int* to be very exact but good enough for our purposes)

From "Arrays and Pointers" by Anand George, SourceLens.org. Link.

So..

- C pointer notation vs. array notation:
  - array[0] is equal to *(array + 0) or *(array)
  - array[1] is equal to *(array + 1)
  - array[2] is equal to *(array + 2)
  - …
- In C pointer math, the units are *elements*
- In MIPS memory accesses, the units are *bytes*
- If `array` lives at 0x1000, then:
  - `array[0]` is `*(array+0)`, and it lives at 0x1000+0*4 = 0x1000
  - `array[1]` is `*(array+1)`, and it lives at 0x1000+1*4 = 0x1004
  - `array[2]` is `*(array+2)`, and it lives at 0x1000+2*4 = 0x1008
  - …

Adapted from "Arrays and Pointers" by Anand George, SourceLens.org. Link.

Pointers vs. arrays

- How they're similar:
  - Both are represented by a memory address which has one or more bytes of content at it
- How they differ:
  - Pointers store a memory address in memory. Only allocated one word (4 bytes on MIPS)
    - Global:

```
char* name=NULL; →  .data
                    name: .word 0
```

    - Local:

```
char* name=NULL; →  addiu $sp, $sp, -16 ; assuming 3 other words needed
                    sw $0, 12($sp)
```

  - Arrays allocated enough space for the array itself. Array declaration by itself doesn't store a memory *address* into memory, compiler just knows where the array lives and makes references to it use that memory region.
    - Global:

```
char name[40]=""; → .data
                    name: .space 40 ; allocates 40 bytes, all null
```

    - Local:

```
char name[40]=""; → addiu $sp, $sp, -52 ; assuming 3 other words needed
                    sb $0, 12($sp) ; only need to set first byte to null
```

What about string literals?

- Pointers to a string literal: put string in read-only data region, store address to it.
  - Global:

```
char* name="hello"; →     .data
                          name: .word str_hello
                          .rdata
                          str_hello: .asciiz "hello"
```

  - Local:

```
char* name="hello"; →     addiu $sp, $sp, -16 ; assuming 3 other words needed
                          la $t0, str_hello
                          sw $t0, 12($sp)
```

`la` is "Load address": a pseudo-isn to put an address into a register.

- Arrays set to string literal: Allocate space for the string and initialize it.
  - Global:

```
char name[40]="hello"; →  .data
                          name: .asciiz "hello"
                          .space 34
```

  - Local:

```
char name[40]="hello"; →  addiu $sp, $sp, -52 ; assuming 3 other words needed
                          add $a0, $sp, 12 ; destination for copy
                          la $a1, str_hello ; source for copy
                          jal memset ; std function to copy memory
```


Pointers vs. arrays

- When you *pass* an array to a function or store a *reference* to an array, then you're using pointers.

```
int func(int* ar) {...}

int my_array[50];
func(my_array);
```

- `ar` gets the address of the start of `my_array`
- Still true for this syntax:

```
int func(int ar[]) {...}
int func(int ar[40]) {...}
```

Multidimensional Arrays

- Again contiguous chunk of data.
- Behaves like
  - array of arrays in the case of 2 dimensions.
  - array of array of array in 3 dimension and so on.
- All are different interpretations and syntaxes on single chunk of contiguous memory

From "Arrays and Pointers" by Anand George, SourceLens.org. Link.

Multidimensional Arrays syntax

```
int mytable[10][20];
```

- We have 10 arrays of array of 20 integers.
- So total 10*20*4 = 800 bytes of contiguous memory.
- As before, the array name acts as a memory address to the beginning of the array.
- Type of the pointer is sort of like `int**`, but the row size needs to be known, so it's actually `int (*mytable)[20]`
- **How to access?** Pointer arithmetic.

`myarray[x][y]`
is equal to
`*(myarray+x*20+y)`

Adapted from "Arrays and Pointers" by Anand George, SourceLens.org. Link.

2D arrays in MIPS

- Global:

```
int a[10][20];
int b[10][20] = {{1,2,3,...,20},{101,102,...,120},...}
```

→

```
.data
a: .space 800
b: .word 1,2,3,4,...,18,19,20,101,...
```

- Local:

```
int a[10][20];
int b[10][20] = {{1,2,3,...,20},{101,102,...,120},...}
```

→

```
addiu $sp, $sp, -1612 ; assuming 3 other words needed
; no initialization for a, so it has whatever trash was on the stack already
li $t0,1
sw $t0, 12($sp)
li $t0,2
sw $t0, 16($sp)
li $t0,3
sw $t0, 20($sp)
  ... ; lots of initialization code
```


What about `sizeof`?

- **`sizeof`** tells you how big something is, in bytes

- For variables declared as **arrays**, this is the size of the array:

```
int array[10]; // sizeof(array) is 10*4 = 40 bytes
int array2d[10][20] // sizeof(array2d) is 10*20*4=800
```

- For pointers, this is the size of one pointer:

```
int* p; // sizeof(p) is 4 bytes
        // (assuming we're on a 32-bit system)
```

- This is true even if the pointer is used to refer to an array:

```
p = array; // p gets the address of the array.
           // sizeof(p) is still 4 bytes
```

Summary

- MIPS ISA and Assembly Programming