EECS 314
Computer Architecture

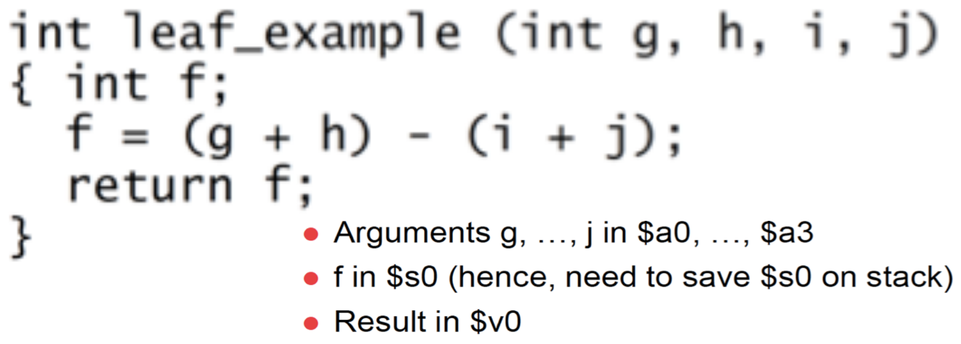
Spring 2018

In age of the C

Instructor: Ming-Chun Huang, PhD

ming-chun.huang@case.edu

Office: Glennan 514B



```
int leaf_example (int g, h, i, j)
{ int f;
  f = (g + h) - (i + j);
  return f;
}
```

- Arguments g, …, j in $a0, …, $a3
- f in $s0 (hence, need to save $s0 on stack)
- Result in $v0

```
jal   ProcedureAddress     #jump and link
```

- Addresses of x, y in $a0, $a1
- i in $s0

e.g. x = an empty space
y = “architecture”

- American Std Code for Info Interchange (ASCII): 8-bit bytes representing characters

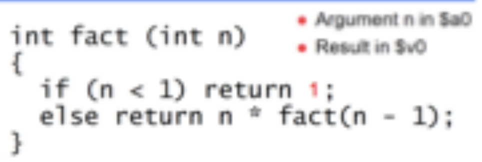
ASCII	Char	ASCII	Char	ASCII	Char	ASCII	Char	ASCII	Char	ASCII	Char
0	Null	32	space	48	0	64	@	96	`	112	p
1		33	!	49	1	65	A	97	a	113	q
2		34	“	50	2	66	B	98	b	114	r
3		35	#	51	3	67	C	99	c	115	s
4	EOT	36	$	52	4	68	D	100	d	116	t
5		37	%	53	5	69	E	101	e	117	u
6	ACK	38	&	54	6	70	F	102	f	118	v
7		39	‘	55	7	71	G	103	g	119	w
8	bksp	40	(	56	8	72	H	104	h	120	x
9	tab	41	)	57	9	73	I	105	i	121	y
10	LF	42	*	58	:	74	J	106	j	122	z
11		43	+	59	;	75	K	107	k	123	{
12	FF	44	,	60	<	76	L	108	l	124	|
15		47	/	63	?	79	O	111	o	127	DEL

e.g. 5! = 5x4x3x2x1=120

```
int fact (int n)
{
  if (n < 1) return 1;
  else return n * fact(n - 1);
}
```

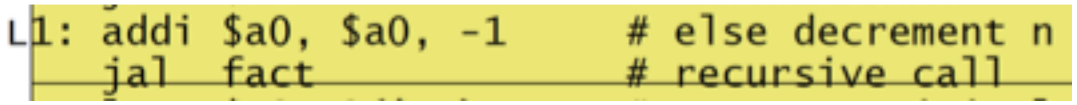
- Argument n in $a0
- Result in $v0

e.g. 5! = 5x4x3x2x1=120



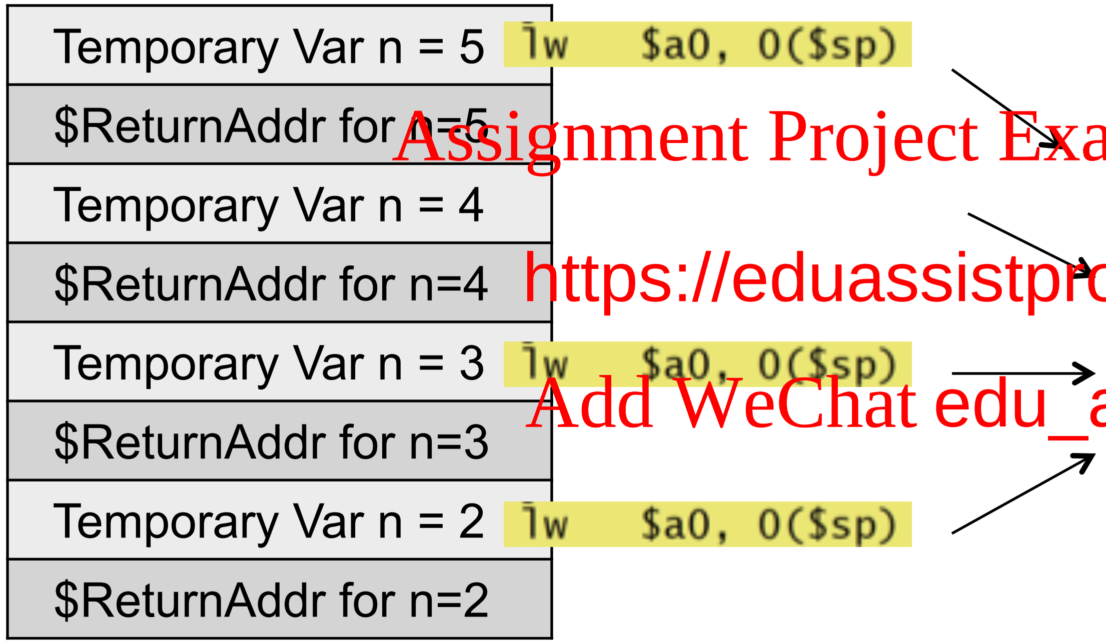
```
int fact (int n)
{
  if (n < 1) return 1;
  else return n * fact(n - 1);
}
```

- Argument n in $a0
- Result in $v0



```
L1: addi $a0, $a0, -1     # else decrement n
    jal  fact             # recursive call
```

Stack in Memory eventually

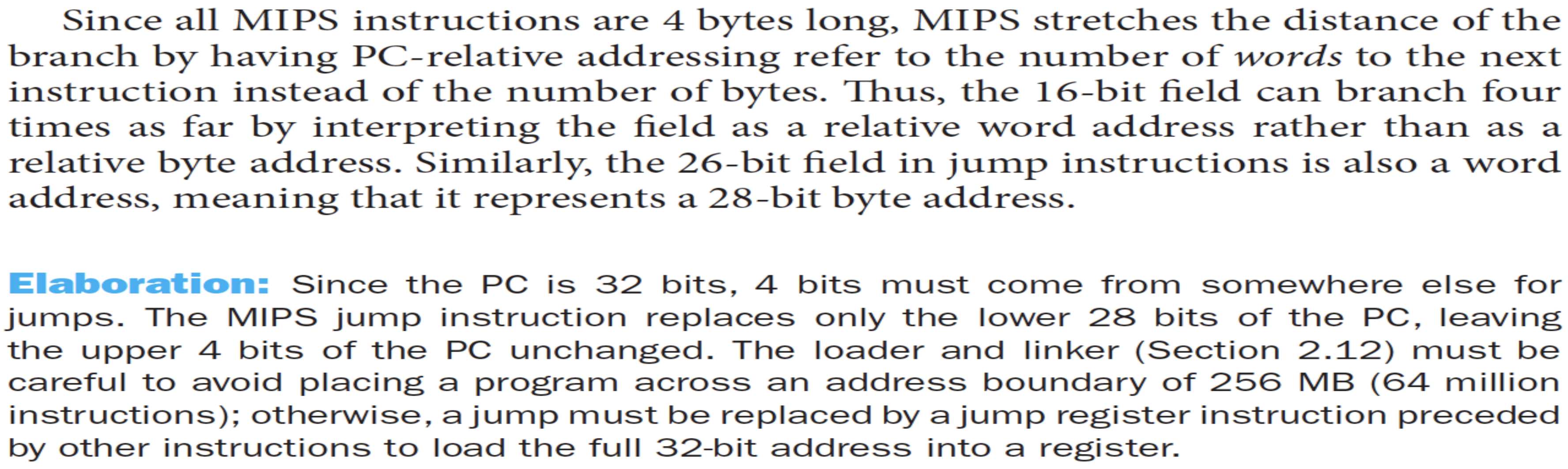


$v0 = 1, initially

e.g. 5! = 5x4x3x2x1=120

```
int fact (int n)
{
  if (n < 1) return 1;
  else return n * fact(n - 1);
}
```

- Argument n in $a0
- Result in $v0



Since all MIPS instructions are 4 bytes long, MIPS stretches the distance of the branch by having PC-relative addressing refer to the number of *words* to the next instruction instead of the number of bytes. Thus, the 16-bit field can branch four times as far by interpreting the field as a relative word address rather than as a relative byte address. Similarly, the 26-bit field in jump instructions is also a word address, meaning that it represents a 28-bit byte address.

Elaboration: Since the PC is 32 bits, 4 bits must come from somewhere else for jumps. The MIPS jump instruction replaces only the lower 28 bits of the PC, leaving the upper 4 bits of the PC unchanged. The loader and linker (Section 2.12) must be careful to avoid placing a program across an address boundary of 256 MB (64 million instructions); otherwise, a jump must be replaced by a jump register instruction preceded by other instructions to load the full 32-bit address into a register.

Here is a traditional loop in C:

```
while (save[i] == k)
      i += 1;
```

Assume that `i` and `k` correspond to registers `$s3` and `$s5` and the base of the array `save` is in `$s6`. What is the MIPS assembly code corresponding to thi C segment?

Remember that MIPS instructions have byte addresses, so addresses of sequential words differ by 4, the number of bytes in a word. The `bne` instruction on the fourth line adds 2 words or 8 bytes to the address of the *following* instruction (80016), specifying the branch destination relative to that following instruction (8 + 80016) instead of relative to the branch instruction (12 + 80012) or using the full destination address (80024). The jump instruction on the last line does use the full address (20000 × 4 = 80000), corresponding to the label `Loop`.

Multiplied by 4