第10回　計算機構成

前々回の内容

- 関数呼出し（続き）
- 再帰関数
  - 階乗を求める関数fact
  - スタック
- 配布物
  - fact関数のトレース

今回の内容

- メール送受信の確認作業
- 中間試験の答合わせ
  - トレースの返却
  - 講評は来週
- R01Qtspim.pdfの配布について
  - Githubからダウンロードできる.

これまでのまとめ

- MIPS命令セット
  - 算術命令 add,addi,sub
  - データ転送 lw,sw
  - 論理演算 and,andi,or,ori,nor
    - ビット演算，ctype.h
  - 条件分岐命令 beq,bne
  - 無条件分岐命令 j
  - 比較命令 slt,slti
  - 関数呼出し jal,$ra, jr
- C言語からMIPSアセンブリ言語への変換
  - if-then-else文
  - while文
  - case/switch文
  - sltu命令により境界チェックの簡便法
  - 関数呼出し
    - jal命令と$raレジスタ，jr命令

末尾再帰（tail recursion）と繰り返し（103頁）

■ 階乗を求める再帰関数

```
int fact(int n)
{
 if( n == 0 ) return 1;
 return n * fact( n-1 ) ;
}

// fact_a( n, 1 )
int fact_a(int n, int v)
{
 if( n == 0 ) return v;
 return fact_a( n-1, v*n );
}
```

■ 再帰から繰り返しに変換

```
int fact_b(int n, int v)
{
  loop:
    if( n==0 ) return v;
    v = v*n ;
    n = n-1 ;
  goto loop;
}

int fact_b(int n, int v)
{
  while( n!=0 )
  {
    v = v*n ;
    n = n-1 ;
  }
  return v;
}
```

MIPSコードに変換せよ.

再帰から繰り返しに変換

■ 合計を求める再帰関数

```
int sum(int n)
{
 if( n == 0 ) return 0;
 return n + sum_a(n-1);
}

// sum_a( n, 0 )
int sum_a(int n, int acc)
{
 if( n == 0 )  return acc;
 return sum_a(n - 1, acc + n);
}
```

■ 再帰から繰り返しに変換

```
int sum_b(int n, int acc)
{
  loop:
     if( n == 0 )  return acc;
     acc = acc + n ;
     n = n -1 ;
  goto loop;
}
```

MIPSコードに変換せよ.

ASCIIコード（105頁）

文字	10進	文字	10進	文字	10進	文字	10進	文字	10進	文字	10進	文字	10進	文字	10進
NUL	0	DLE	16	SP	32	0	48	@	64	P	80	`	96	p	112
SOH	1	DC1	17	!	33	1	49	A	65	Q	81	a	97	q	113
STX	2	DC2	18	"	34	2	50	B	66	R	82	b	98	r	114
ETX	3	DC3	19	#	35	3	51	C	67	S	83	c	99	s	115
EOT	4	DC4	20	$	36	4	52	D	68	T	84	d	100	t	116
ENQ	5	NAK	21	%	37	5	53	E	69	U	85	e	101	u	117
ACK	6	SYN	22	&	38	6	54	F	70	V	86	f	102	v	118
BEL	7	ETB	23	'	39	7	55	G	71	W	87	g	103	w	119
BS	8	CAN	24	(	40	8	56	H	72	X	88	h	104	x	120
HT	9	EM	25	)	41	9	57	I	73	Y	89	i	105	y	121
NL*	10	SUB	26	*	42	:	58	J	74	Z	90	j	106	z	122
VT	11	ESC	27	+	43	;	59	K	75	[	91	k	107	{	123
NP	12	FS	28	,	44	<	60	L	76	¥	92	l	108	|	124
CR	13	GS	29	-	45	=	61	M	77	]	93	m	109	}	125
SO	14	RS	30	.	46	>	62	N	78	^	94	n	110	~	126
SI	15	US	31	/	47	?	63	O	79	_	95	o	111	DEL	127

バイト転送命令と文字列コピー手続き

■ バイト転送命令

```
lb $t0,0($sp)
sb $t0,0($sp)
```

■ 文字列コピー

```
void strcpy( char x[], char y[])
{
   int i; // $s0
   i = 0;
   while( (x[i]=y[i]) != 0 )
      i = i + 1;
}
```

"ABCD"

41	42	43	44	0

■ MIPSコード

```
strcpy:
   addi $sp,$sp,-4
   sw $s0,4($sp)
   add $s0,$zero,$zero
L1:add $t1,$a1,$s0
   lb $t2,0($t1)
   add $t3,$a0,$s0
   sb $t2,0($t3)
   beq $t2,$zero,L2
   add $s0,$s0,1
   j L1
L2:lw $s0,4($sp)
   addi $sp,$sp,4
   jr $ra
```

効率の良いMIPSコード

■MIPSコード

```
strcpy:
  addi $sp,$sp,-4
  sw $s0,4($sp)
  add $s0,$zero,$zero
L1:add $t1,$a1,$s0
  lb $t2,0($t1)
  add $t3,$a0,$s0
  sb $t2,0($t3)
  beq $t2,$zero,L2
  add $s0,$s0,1
  j L1
L2:lw $s0,4($sp)
  addi $sp,$sp,4
  jr $ra
```

■ 効率の良いMIPSコード

```
strcpy:
   addi $sp,$sp,-4
   sw $s0,4($sp)
   add $s0,$zero,$zero
   beq $zero,$zero,L2
L1:add $s0,$s0,1
L2:add $t1,$a1,$s0
   lb $t2,0($t1)
   add $t3,$a0,$s0
   sb $t2,0($t3)
   bne $t2,$zero,L1
   lw $s0,4($sp)
   addi $sp,$sp,4
   jr $ra
```

数字の文字列を数値に変換する

■ 再帰関数

```
int num( char *p, int n )
{
    if ( *p == '\0')  return n ;
    return num(p+1, n*10 + *p-'0');
}
```

■ 繰り返し

```
int num( char *p )
{
  int n ;
  n = 0 ;
  while( *p != '\0' )
  {
    n = n * 10 + ( *p - '0' );
    p++;
  }
  return n ;
}
```

lui命令　load upper immediate（109頁）

■ レジスタ上位１６ビットに値を設定する命令

- 命令で指定した１６ビットの即値をレジスタの左側にロードし，右側を０で埋める

■ lui $t0,255

001111 00000　01000 0000 0000 1111 1111

$t0:0000000011111111 0000000000000000

３２ビット即値のロード

■ $s0 = 0000 0000 0011 1101 0000 1001 0000 0000

```
lui $s0,61        #    61dec = 0000 0000 0011 1101
ori $s0,$s0,2034  # 2034dec = 0000 1001 0000 0000

$s0: 0000 0000 0011 1101 0000 0000 0000 0000
$s0: 0000 0000 0011 1101 0000 1001 0000 0000
```

■ oriの動作　１６ビットの即値は上位ビットが０が埋められる

```
          0000 0000 0011 1101 0000 0000 0000 0000
論理和    0000 0000 0000 0000 0000 1001 0000 0000
$s0:      0000 0000 0011 1101 0000 1001 0000 0000
```

論理演算では，符号拡張はしてはいけない

j命令とjal命令　擬似直接アドレッシング

■ １命令は４バイト固定である
→ジャンプ先アドレスは４番地おきである
→ジャンプ先アドレスの下位２ビットは必ず０
→アドレスフィールドには４分の１の値を格納すればよい

■ 10000番地に無条件ジャンプ

j 2500

2	2500
6bits	26bits

■ 10000番地にジャンプ＆リンク

jal 2500

3	2500
6bits	26bits

- $ra=PC+4; goto 10000

■ 擬似直接アドレッシング

- MIPSのアドレスは32ビット
- 命令では26+2ビットしか指定していない
- 4ビット不足している
  - 不足しているので「擬似」

■ PC=JumpAddr

JumpAddr={**PC{31:28}**,address,2b'0}

4ビット　26ビット　2ビット

bne/beq命令　PC相対アドレッシング

■ １命令は４バイト固定である
→相対番地でも４バイト単位で分岐する
→アドレスフィールドには４分の１の値を格納すればよい

■ 条件分岐

bxx $s0,$s1,Exit

5	16	17	Exit
6bits	5bits	5bits	16bits

■ 条件が成立したとき　PC=PC+4+(-2^{15}~2^{15}-1)×4

条件が成立しないとき PC=PC+4

■ $s1と$s2が等しいとき，PC相対で100番地先に飛ぶ

- beq $s1,$s2,25 #if($s1==$s2)goto PC+4+100

■ "PC+4"は何か？

- PC=**PC+4**+(-2^{15}~2^{15}-1)×4
- PC=**PC+4**

■ "PC+4"は次の命令のアドレスである

whileループ

■ Cコード

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

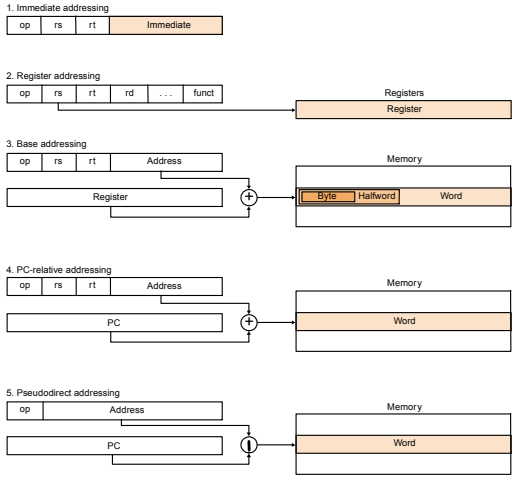
- i,kは$s3,$s5
- saveのベースアドレスは$s6

■ MIPSコード

```
Loop:sll $t1,$s3,2
     add $t1,$t1,$s6
     lw $t0,0($t1)
     bne $t0,$s5,Exit
     addi $s3,$s3,1
     j Loop
Exit:
```

080000	0	0	19	9	2	0	sll $t1,$s3,2
080004	0	9	22	9	0	32	add $t1,$t1,$s6
080008	35	9	8	0			lw $t0,0($t1)
080012	5	8	21	2			bne $t0,$s5,Exit
080016	8	19	19	1			addi $s3,$s3,1
080020	2	2000					j Loop
080024							

MIPSアドレシングモード



より遠くへの分岐

■ L1の値が16ビットに収まらないとき

```
beq $s0,$s1,L1
```

■ j命令と組み合わせれば良い.

```
    bne $s0,$s1,L2
    j L1
L2:
```

オーバーフローの検出（208頁）

■ 符号付き加算でのオーバーフロー

```
    addu $t0,$t1,$t2
    xor $t3,$t1,$t2
    slt $t3,$t3,$zero
    bne $t3,$zero,NoOV
    xor $t3,$t0,$t1
    slt $t3,$t3,$zero
    bne $t3,$zero,OV
NoOV: //オーバーフローなし

OV:   //オーバーフロー処理
```

■ 符号無し加算でのオーバーフロー

```
    addu $t0,$t1,$2
    nor $t3,$t1,$zero
    sltu $t3,$t3,$t2
    bne $t3,$zero,OV
NoOV: //オーバーフローなし

OV://オーバーフロー処理
```

オーバフロー例外を発生させないため
add　ではなくて　addu
を使う.

オーバーフロー　４ビット符号付き整数同士の加算

- 異符号同士の加算では発生しない
- 同符号同士の加算で発生する場合がある．
  - 正＋正＝負
  - 負＋負＝正

		-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
		1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111
-8	1000	OV	OV	OV	OV	OV	OV	OV	OV	1000	1001	1010	1011	1100	1101	1110	1111
-7	1001	OV	OV	OV	OV	OV	OV	OV	1000	1001	1010	1011	1100	1101	1110	1111	0000
-6	1010	OV	OV	OV	OV	OV	OV	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001
-5	1011	OV	OV	OV	OV	OV	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010
-4	1100	OV	OV	OV	OV	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011
-3	1101	OV	OV	OV	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100
-2	1110	OV	OV	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101
-1	1111	OV	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110
0	0000	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111
1	0001	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	OV
2	0010	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	OV	OV
3	0011	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	OV	OV	OV
4	0100	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	OV	OV	OV	OV
5	0101	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	OV	OV	OV	OV	OV
6	0110	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	OV	OV	OV	OV	OV	OV
7	0111	1111	0000	0001	0010	0011	0100	0101	0110	0111	OV	OV	OV	OV	OV	OV	OV

符号付き加算におけるオーバーフローの検出

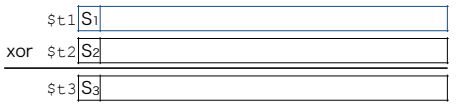
- 符号付き加算でのオーバーフロー

```
addu $t0,$t1,$t2
xor  $t3,$t1,$t2
slt  $t3,$t3,$zero
bne  $t3,$zero,NoOV
xor  $t3,$t0,$t1
slt  $t3,$t3,$zero
bne  $t3,$zero,OV
```

NoOV: //オーバーフローなし

OV:　//オーバーフロー処理

- $t1と$t2が異符号ならばオーバーフローしない
- xorで同符号か異符号かを調べることができる
  - $t1と$t2が同符号ならば S_3=0となり，$t3は0以上の正数である
  - $t1と$t2が異符号ならば S_3=1となり，$t3は負数である



- $t1と$t2が同符号ならば，加算結果$t0と$t1（$t2）の符号が異なればオーバーフローが発生している．

オーバーフロー　４ビット符号無し整数同士の加算

- 加算の結果が$(1111)_2$を超えるとき
- 1011+0110=10001
- 検出条件
  - X+Y>10000

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0001	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	OV
0010	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	OV	OV
0011	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	OV	OV	OV
0100	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	OV	OV	OV	OV
0101	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	OV	OV	OV	OV	OV
0110	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	OV	OV	OV	OV	OV	OV
0111	0111	1000	1001	1010	1011	1100	1101	1110	1111	OV	OV	OV	OV	OV	OV	OV
1000	1000	1001	1010	1011	1100	1101	1110	1111	OV	OV	OV	OV	OV	OV	OV	OV
1001	1001	1010	1011	1100	1101	1110	1111	OV	OV	OV	OV	OV	OV	OV	OV	OV
1010	1010	1011	1100	1101	1110	1111	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV
1011	1011	1100	1101	1110	1111	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV
1100	1100	1101	1110	1111	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV
1101	1101	1110	1111	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV
1110	1110	1111	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV
1111	1111	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV

符号無し加算におけるオーバーフローの検出

- 符号無し加算でのオーバーフロー

```
addu $t0,$t1,$2
nor  $t3,$t1,$zero
sltu $t3,$t3,$t2
bne  $t3,$zero,OV
```

NoOV: //オーバーフローなし

OV://オーバーフロー処理

- ビット反転（NOT）はnor命令で計算できる

```
nor $t3,$t1,$zero # $t3= NOT $t1
```

- ビット反転した値は，次のように表すことができる

```
$t3= NOT $t1 = 2^32-1 - $t1
```

- sltu $t3,$t3,$t2の意味は次の通り．

```
$t3          < $t2
2^32-1 - $t1 < $t2
2^32-1       < $t2+$t1
```

32ビットの最大値

- 次の条件が成立すれば，オーバーフローが発生している

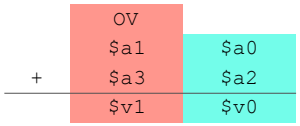
```
2^32-1 < $t2+$t1
```

2倍長の加算 dadd.s

- 64ビット同士の加算をする
- 上位32ビットと下位32ビットに分けて考える
  - 下位32ビットは符号無し加算で，オーバーフローを検出する
  - 上位32ビットは符号付き加算で，下位32ビットの加算からオーバーフローがあれば加算結果に 1 を加える．

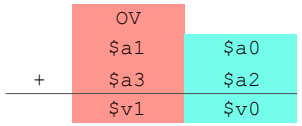
```
  .text
  .globl __start
__start:
   add $a0,$zero,0x40000000
   add $a1,$zero,0x00001234
   add $a2,$zero,0x3FFFFFFF
   add $a3,$zero,0x0
   jal dadd
   break 2
```

```
dadd: add $v1,$zero,$zero
      addu $v0,$a0,$a2
      nor $t1,$a0,$zero
      sltu $t1,$t1,$a2
      beq $t1,$zero,No_OV
      add $v1,$v1,1
No_OV:add $v1,$v1,$a1
      add $v1,$v1,$a3
      jr $ra
```



2倍長の加算 dadd.s

- 64ビット同士の加算をする
- 上位32ビットと下位32ビットに分けて考える
  - 下位32ビットは符号無し加算で，オーバーフローを検出する
  - 上位32ビットは符号付き加算で，下位32ビットの加算からオーバーフローがあれば加算結果に 1 を加える．




```
      .text
      .globl __start
__start:
      add $a0,$zero,0x40000000
      add $a1,$zero,0x00001234
      add $a2,$zero,0x3FFFFFFF
      add $a3,$zero,0x0
      jal dadd
      break 2
dadd: add $v1,$zero,$zero
      addu $v0,$a0,$a2
      nor $t1,$a0,$zero
      sltu $t1,$t1,$a2
      beq $t1,$zero,No_OV
      add $v1,$v1,1
No_OV:add $v1,$v1,$a1
      add $v1,$v1,$a3
      jr $ra
```