

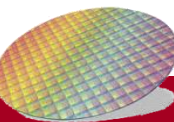


成功大學

National Cheng Kung University

Computer Organization Lab1 MIPS Simulator - SPIM

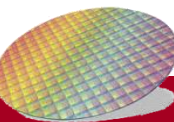
ICES Lab





QtSPIM

- QtSPIM 是個簡易的MIPS Simulator
- 擁有Assembler，可將MIPS code轉換成machine code並執行
- 可output 編過的machine code file
- 介面含三個部分：
 - Register File
 - Text Segment
 - Data Segment



QtSPIM



清华大学

Register file

Text Segment

QtSpim

File Simulator Registers Text Segment Data Segment Window Help

FP Regs Int Regs [16] Data Text

Int Regs [16]

```
PC = 0
EPC = 0
Cause = 0
BadVAddr = 0
Status = 3000fff10

HI = 0
LO = 0

R0 [r0] = 0
R1 [at] = 0
R2 [v0] = 0
R3 [v1] = 0
R4 [a0] = 3
R5 [a1] = 7ffff484
R6 [a2] = 7ffff494
R7 [a3] = 0
R8 [t0] = 0
R9 [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
```

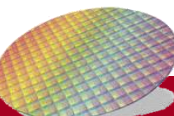
Text

```
User Text Segment [00400000]..[00440000]
[00400000] 8fa40000 lw $4, 0($29) ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004 addiu $5, $29, 4 ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004 addiu $6, $5, 4 ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080 sll $2, $4, 2 ; 186: sll $v0 $a0 2
[00400010] 00c23021 addu $6, $6, $2 ; 187: addu $a2 $a2 $v0
[00400014] 0c000000 jal 0x00000000 [main] ; 188: jal main
[00400018] 00000000 nop ; 189: nop
[0040001c] 3402000a ori $2, $0, 10 ; 191: li $v0 10
[00400020] 0000000c syscall ; 192: syscall # syscall 10 (exit)

Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821 addu $27, $0, $1 ; 90: move $k1 $at # Save $at
[80000184] 3c019000 lui $1, -28672 ; 92: sv $v0 s1 # Not re-entrant and we can't trust $sp
[80000188] ac220200 sw $2, 512($1)
[8000018c] 3c019000 lui $1, -28672 ; 93: sv $a0 s2 # But we need to use these registers
[80000190] ac240204 sw $4, 516($1)
[80000194] 401a6800 mfc0 $26, $13 ; 95: mfc0 $k0 $13 # Cause register
[80000198] 001a2082 srl $4, $26, 2 ; 96: srl $a0 $k0 2 # Extract ExcCode Field
[8000019c] 3084001f andi $4, $4, 31 ; 97: andi $a0 $a0 0x1f
[800001a0] 34020004 ori $2, $0, 4 ; 101: li $v0 4 # syscall 4 (print_str)
[800001a4] 3c049000 lui $4, -28672 [__m1_] ; 102: la $a0 __m1_
[800001a8] 0000000c syscall ; 103: syscall
[800001ac] 34020001 ori $2, $0, 1 ; 105: li $v0 1 # syscall 1 (print_int)
[800001b0] 001a2082 srl $4, $26, 2 ; 106: srl $a0 $k0 2 # Extract ExcCode Field
[800001b4] 3084001f andi $4, $4, 31 ; 107: andi $a0 $a0 0x1f
[800001b8] 0000000c syscall ; 108: syscall
[800001bc] 34020004 ori $2, $0, 4 ; 110: li $v0 4 # syscall 4 (print_str)
[800001c0] 3344003c andi $4, $26, 60 ; 111: andi $a0 $k0 0x3c
[800001c4] 3c019000 lui $1, -28672 ; 112: lw $a0 __excpc($a0)
[800001c8] 00240821 addu $1, $1, $4
[800001cc] 8c240180 lw $4, 384($1)
[800001d0] 00000000 nop ; 113: nop
[800001d4] 0000000c syscall ; 114: syscall
[800001d8] 34010018 ori $1, $0, 24 ; 116: bne $k0 0x18 ok_pc # Bad PC exception requires
```

Memory and registers cleared

SPIM Version 9.1.17 of January 1, 2016
Copyright 1990-2016, James R. Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and version 2.1.





QtSPIM

Data Segment

The screenshot displays the QtSPIM application window. The 'Data' tab is selected, showing the 'User data segment' and 'User Stack'. The 'User data segment' contains a string 'Hello World'. The 'User Stack' contains a series of memory addresses and their corresponding values. The 'Registers' window on the left shows the state of various registers, including PC, EPC, Cause, BadVAddr, Status, HI, LO, R0-R15, R16-R31, and R27-R31. The bottom status bar indicates 'Memory and registers cleared' and provides version information for SPIM and QtSPIM.

QtSPIM

File Simulator Registers Text Segment Data Segment Window Help

FP Regs Int Regs [16] Data Text

Int Regs [16]

PC = 400020
EPC = 0
Cause = 0
BadVAddr = 0
Status = 3000ff10

HI = 0
LO = 0

R0 [r0] = 0
R1 [at] = 0
R2 [v0] = a
R3 [v1] = 0
R4 [a0] = 10010000
R5 [a1] = 7ffff484
R6 [a2] = 7ffff494
R7 [a3] = 0
R8 [t0] = 0
R9 [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0

Data

User data segment [10000000]..[10040000]

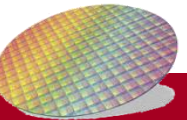
[10000000]..[1000ffff] 00000000
[10010000] 6c6c6548 6f57206f 00646c72 00000000 H e l l o W o r l d
[10010010]..[1003ffff] 00000000

User Stack [7ffff480]..[80000000]

[7ffff480] 00000003 7ffff576 7ffff570 7ffff556 v . . . p . . . V . . .
[7ffff490] 00000000 7ffff5e1 7ffff5b7 7ffff580
[7ffff4a0] 7ffff544 7ffff513 7ffff5db 7ffff5c2 D
[7ffff4b0] 7ffff59e 7ffff577 7ffff52e 7ffff51a w
[7ffff4c0] 7ffff50d 7ffff5f4 7ffff5dc 7ffff5ad
[7ffff4d0] 7ffff496 7ffffd88 7ffff999 7ffff95b
[7ffff4e0] 7ffff927 7ffff90c 7ffff8ef 7ffff8a7 '
[7ffff4f0] 7ffff895 7ffff87d 7ffff862 7ffff83e } . . . b . . . > . . .
[7ffff500] 7ffff815 7ffff7f7 7ffff7b6 7ffff79f
[7ffff510] 7ffff780 7ffff76c 7ffff759 7ffff74a l . . . Y . . . J . . .
[7ffff520] 7ffff734 7ffff70a 7ffff6e1 7ffff6bf 4
[7ffff530] 7ffff68a 7ffff673 7ffff661 7ffff643 s . . . a . . . C . . .
[7ffff540] 7ffff5f1 7ffff5df 7ffff5c7 7ffff581
[7ffff550] 00000000 78280000 2f293638 70537451 (x 8 6) / Q t S p
[7ffff560] 682f6d69 6f6c6c65 6c726f77 00732e64 i m / h e l l o w o r l d . s .
[7ffff570] 656c6946 3a430073 6f72502f 6d617267 F i l e s . C : / P r o g r a m
[7ffff580] 6e697700 73776f64 6172745f 676e6963 . w i n d o w s _ t r a c i n g
[7ffff590] 676f6c5f 656c6966 5c3a433d 42545642 _ l o g f i l e = C : \ B V T B
[7ffff5a0] 545c6e69 73747365 736e695c 6c6c6174 i n \ T e s t s \ i n s t a l l
[7ffff5b0] 6b636170 5c656761 6c697363 6966676f p a c k a g e \ c s i l o g f i
[7ffff5c0] 6c2e656c 7700676f 6f646e69 745f7377 l e . l o g . w i n d o w s _ t
[7ffff5d0] 69636172 665f676e 7367616c 7700333d r a c i n g _ f l a g s = 3 . w
[7ffff5e0] 69646e69 3a433d72 6e69575c 73776f64 i n d i r = C : \ W i n d o w s
[7ffff5f0] 31535600 4f433031 4f544e4d 3d534c4f . V S 1 1 0 C O M M T O O L S =
[7ffff600] 505c3a43 72676f72 46206d61 73656c69 C : \ P r o g r a m F i l e s
[7ffff610] 38782820 4d5c2936 6f726369 74666f73 (x 8 6) \ M i c r o s o f t
[7ffff620] 73695620 206c6175 64757453 31206f69 V i s u a l S t u d i o 1
[7ffff630] 5c302e31 6d6d6f43 5c376e6f 6c6f6f54 1 . 0 \ C o m m o n 7 \ T o o l
[7ffff640] 55005c73 50524553 49464f52 433d454c s \ . U S E R P R O F I L E = C
[7ffff650] 73555c3a 5c737265 6572654a 6c74796d : \ U s e r s \ J e r e m y t l


Memory and registers cleared

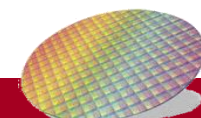
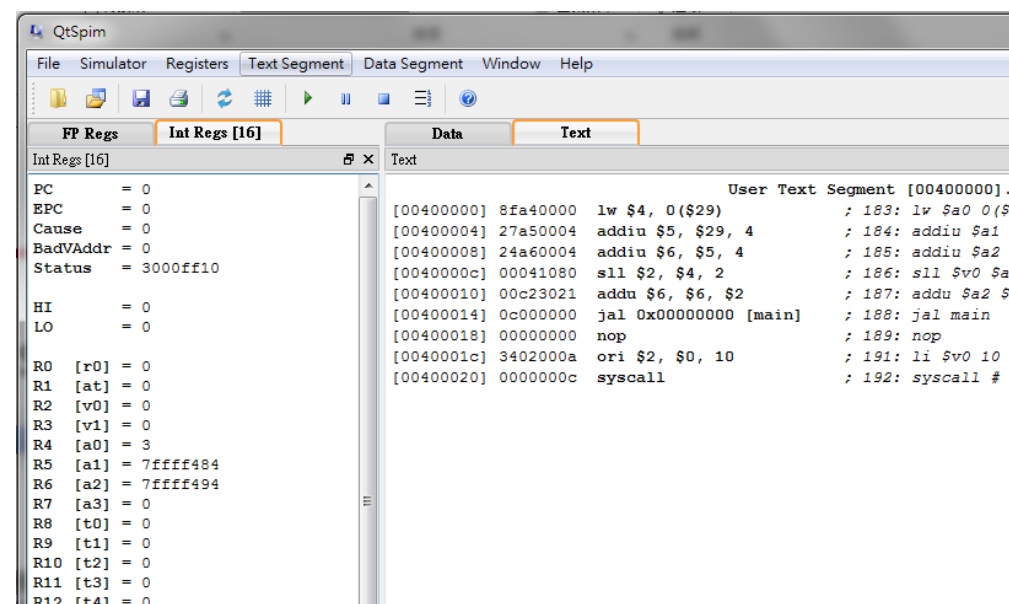
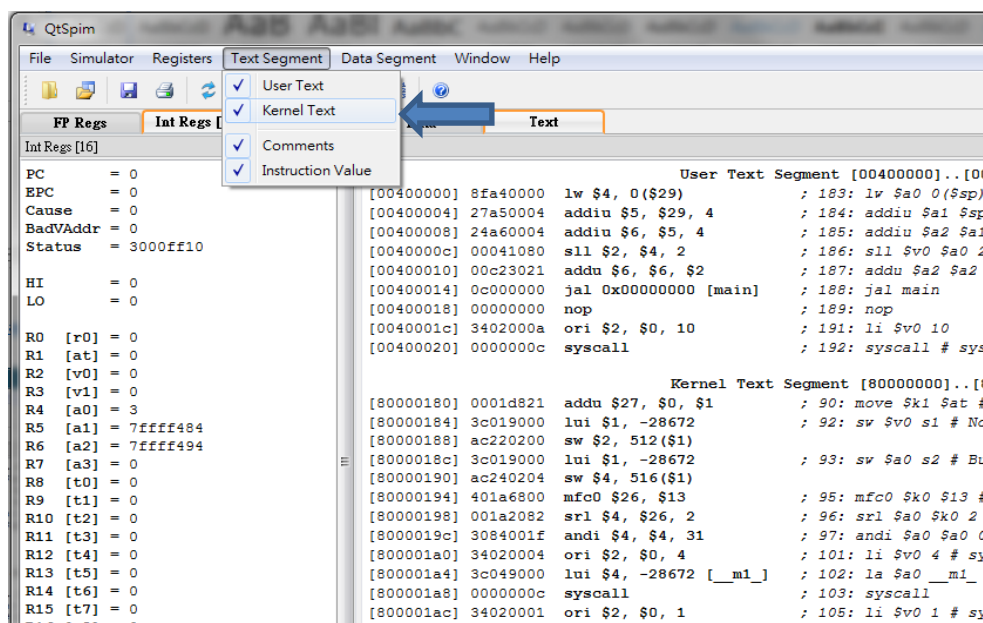
SPIM Version 9.1.17 of January 1, 2016
Copyright 1990-2016, James R. Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and version 2.1.







QtSPIM

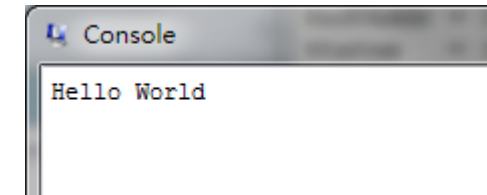
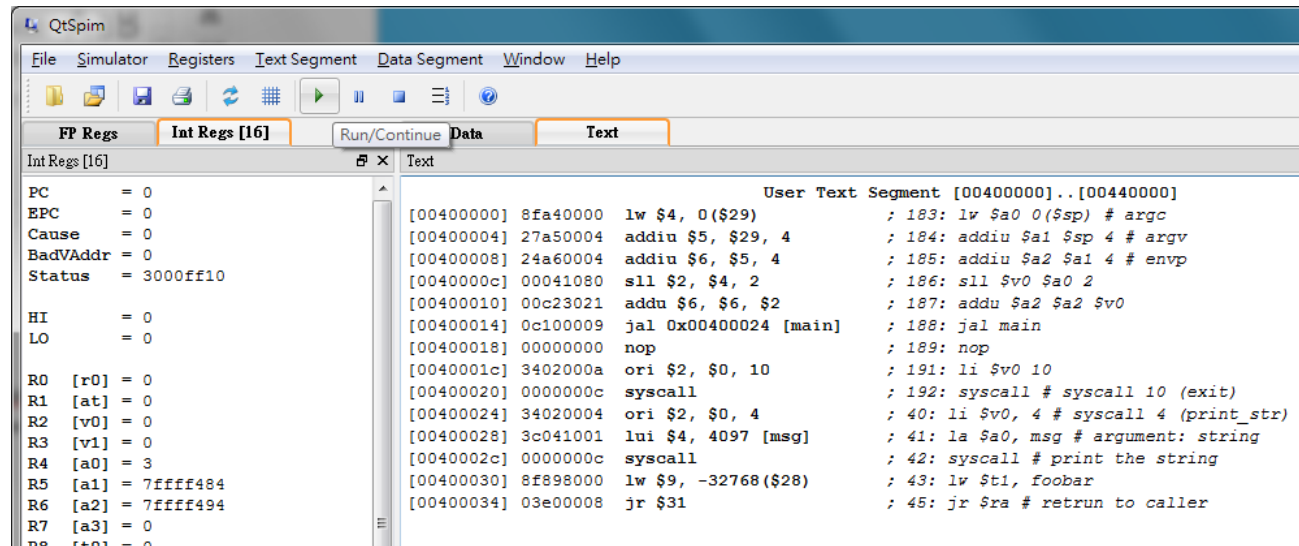
- 環境Setup
 - Text Segment > 取消勾選 Kernel Text
 - 再點  reinitialize simulator



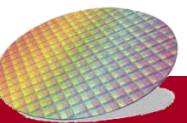


QtSPIM

- Load file
 - 點選  reinitialize and load file，選擇要編的assembly code file
 - 範例:QSPIM資料夾內的helloworld.s
 - 點選  Run/Continue
 - Console output: Hello World



*若出現Attempt to execute non-instruction，直接按OK即可

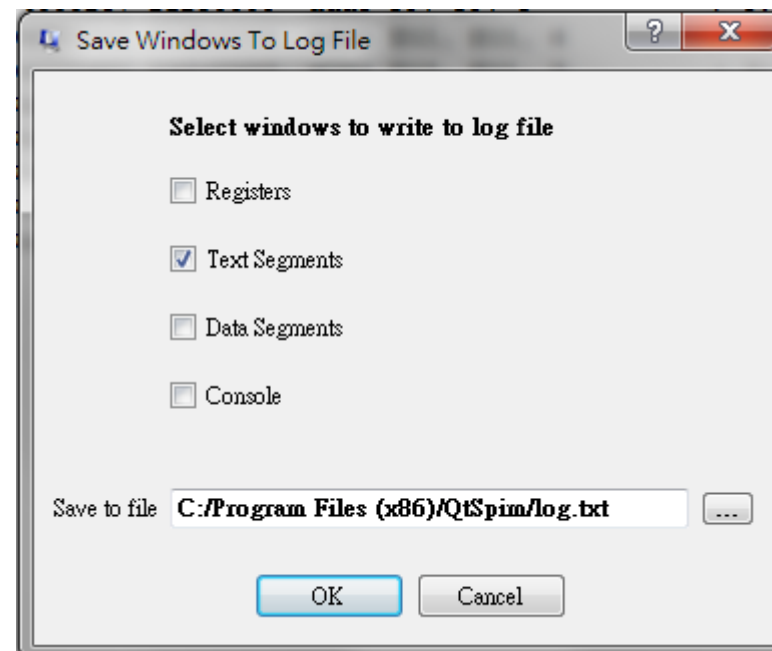




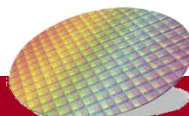
QtSPIM

- Output machine code
 - 點選  勾選Text Segment
 - 選擇要存的folder並命名xxx.txt

```
User Text Segment [00400000]..[00440000]
[00400000] 8fa40000 lw $4, 0($29) ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004 addiu $5, $29, 4 ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004 addiu $6, $5, 4 ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080 sll $2, $4, 2 ; 186: sll $v0 $a0 2
[00400010] 00c23021 addu $6, $6, $2 ; 187: addu $a2 $a2 $v0
[00400014] 0c100009 jal 0x00400024 [main] ; 188: jal main
[00400018] 00000000 nop ; 189: nop
[0040001c] 3402000a ori $2, $0, 10 ; 191: li $v0 10
[00400020] 0000000c syscall ; 192: syscall # syscall 10 (exit)
[00400024] 2108000c addi $8, $8, 12 ; 4: addi $t0, $t0, 12
[00400028] 21290005 addi $9, $9, 5 ; 5: addi $t1, $t1, 5
[0040002c] 216b0004 addi $11, $11, 4 ; 7: addi $t3, $t3, 4
[00400030] 296c0003 slti $12, $11, 3 ; 9: slti $t4, $t3, 3 #if($t3
[00400034] 11800003 beq $12, $0, 12 [LABEL-0x00400034]
[00400038] 01095020 add $10, $8, $9 ; 12: add $t2, $t0, $t1
[0040003c] 08100011 j 0x00400044 [QUIT] ; 13: j QUIT
[00400040] 01095022 sub $10, $8, $9 ; 15: sub $t2, $t0, $t1
[00400044] 03e00008 jr $31 ; 18: jr $ra # retrun to caller
```



PC	Machine code(hex)	assembly	comment
----	----------------------	----------	---------





MIPS instruction

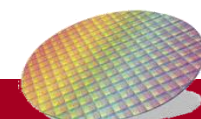
R Type

opcode		rs	rt	rd	shamt	funct					
31	26	25	21	20	16	15	11	10	6	5	0
opcode	Mnemonics	SRC1	SRC2	DST	funct	Description					
000000	nop	00000	00000	00000	000000	No operation					
000000	add	\$Rs	\$Rt	\$Rd	100000	Rd = Rs + Rt					
000000	sub	\$Rs	\$Rt	\$Rd	100010	Rd = Rs − Rt					
000000	and	\$Rs	\$Rt	\$Rd	100100	Rd = Rs & Rt					
000000	or	\$Rs	\$Rt	\$Rd	100101	Rd = Rs Rt					
000000	xor	\$Rs	\$Rt	\$Rd	100110	Rd = Rs ^ Rt					
000000	nor	\$Rs	\$Rt	\$Rd	100111	Rd = ~(Rs Rt)					
000000	slt	\$Rs	\$Rt	\$Rd	101010	Rd = (Rs < Rt)?1:0					
000000	sll		\$Rt	\$Rd	000000	Rd = Rt << shamt					
000000	srl		\$Rt	\$Rd	000010	Rd = Rt >> shamt					
000000	jr	\$Rs			001000	PC=Rs					

Assembly syntax:

op **\$Rd**, \$Rs, \$Rt

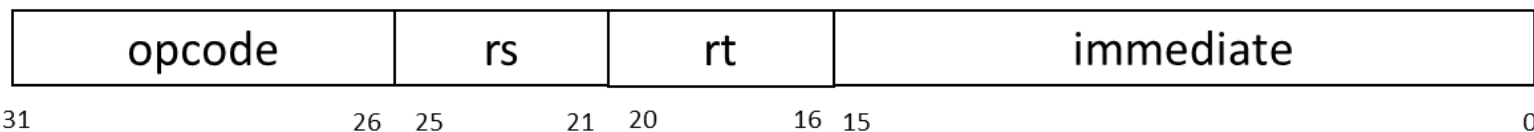
※ jr \$ra (相等於 return)





MIPS instruction

I Type

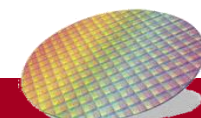


opcode	Mnemonics	SRC1	DST	SRC2	Description
001000	addi	\$Rs	\$Rt	imm	Rt = Rs + imm
001100	andi	\$Rs	\$Rt	imm	Rt = Rs & imm
001010	slti	\$Rs	\$Rt	imm	Rd = (Rs < imm) ? 1 : 0
000100	beq	\$Rs	\$Rt	imm	If(Rs == Rt) PC=PC+4+imm
000101	bne	\$Rs	\$Rt	imm	If(Rs != Rt) PC=PC+4+imm
100011	lw	\$Rs	\$Rt	imm	Rt = Mem[Rs + imm]
101011	sw	\$Rs	\$Rt	imm	Mem[Rs + imm] = Rt

Assembly syntax:

op	\$Rt,	\$Rs,	imm
----	-------	-------	-----

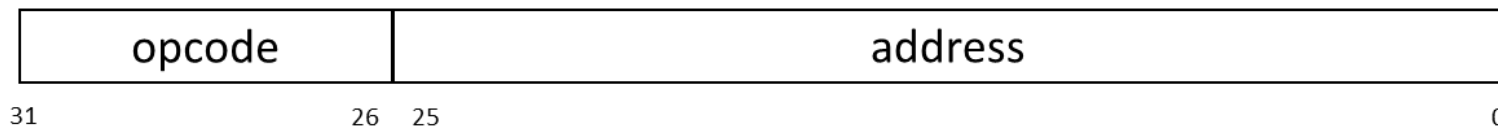
⌘ bne/beq \$Rt, \$Rs, Label





MIPS instruction

J Type

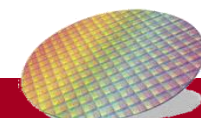


opcode	Mnemonics	SRC1	Description
000010	j	jumpAddr	PC = jumpAddr
000011	jal	jumpAddr	R[31] = PC + 8 ; PC = jumpAddr

※R[31]=\$ra #return address

Assembly Syntax:

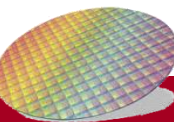
op	jumpAddr/Label
----	----------------





MIPS instruction

- Demo
 - Addition & subtraction
 - Branch
 - Single step
 - Set Breakpoint





Lab1

- 實作出 $1 + 2 + 3 + \dots + 25$ 並將結果存放在 **\$t0**
 - Code中必須有iteration (利用Branch) ，不可直接將正確答案放入\$t0

