國立中正大學106學年度碩士班招生考試試題

信號與媒體通訊組

系所別：電機工程學系-計算機工程組　　科目：計算機組織

晶片系統組

1. (20%) Please explain the terms of following:

   (a). Branch Delay Slot, (b). Datapath, (c). Hazard, (d). Fowarding, (e). Single-cycle CPU

2. (10%) Please resolve the following cache designs:

   2.1 (5%) A caches is designed with 128 blocks and 8 bytes for each block. For the memory byte address 1280, what is the mapped cache block by using direct-map mechanism?

   2.2 (5%) Assume the address format of cache is 32-bit. What is the size for tag field?

3. (20%) By considering the following instructions for a five stages MIPS CPU:

```
lw   $5,  -10 ($5)
sw   $5,  -10 ($5)
sub  $5,   $5, $5
```

   3.1 (10%) By assuming there is not any forwarding in this pipelined processor, please plot the pipeline diagram of instructions with bubbles (pipeline stall).

   3.2 (10%) Again please plot pipeline diagram with forwarding.

4. (6%) A CPU has a clock rate of 4GHz and voltage of 1V. Assume that, on average, it consumes 30W of static power and 40W of dynamic power. If the supply voltage is reduced by 10%, how much percentage of total power saving can be achieved?

5. (15%) The individual stages of a datapath have the following latencies:

IF	ID	EX	MEM	WB
500 ps	400 ps	350 ps	450 ps	300 ps

   5.1 (6%) What are the clock frequencies of a pipelined and non-pipelined processor?

   5.2 (3%) What is the total latency of a **sw** instruction in a non-pipelined processor?

   5.3 (6%) If we can combine two adjacent stages of the pipelined datapath and then split the combined one into three new stages, which stages would you choose and what is the optimum clock cycle time of the new processor?

6. (8%) What's the difference between "response time" and "throughput" of a CPU? Give an idea to improve "response time" and "throughput" of a CPU, respectively.

7. (6%) When designing memory hierarchy in a computer system with DRAM and SRAM, which one is used for cache memory and which one for main memory? Why?

8. (15%) Implement the function "unsigned int Fib (unsigned int n)" which returns the value of the n^{th} Fibonacci number, Fib (0) = 0, Fib (1) = 1, Fib (2) = 1, Fib (3) = 2, ..., Fib (n) = Fib ($n-1$) + Fib ($n-2$).

   8.1 (6%) Write the C code.

   8.2 (9%) Translate your C code into MIPS code. Assume that the argument n is in $a0, and the result is in $v0.

MIPS32® Instruction Set Quick Reference

RD	— DESTINATION REGISTER
RS, RT	— SOURCE OPERAND REGISTERS
RA	— RETURN ADDRESS REGISTER (R31)
PC	— PROGRAM COUNTER
ACC	— 64-BIT ACCUMULATOR
LO, HI	— ACCUMULATOR LOW ($ACC_{31:0}$) AND HIGH ($ACC_{63:32}$) PARTS
±	— SIGNED OPERAND OR SIGN EXTENSION
∅	— UNSIGNED OPERAND OR ZERO EXTENSION
::	— CONCATENATION OF BIT FIELDS
R2	— MIPS32 RELEASE 2 INSTRUCTION
DOTTED	— ASSEMBLER PSEUDO-INSTRUCTION

PLEASE REFER TO *"MIPS32 ARCHITECTURE FOR PROGRAMMERS VOLUME II: THE MIPS32 INSTRUCTION SET"* FOR COMPLETE INSTRUCTION SET INFORMATION.

ARITHMETIC OPERATIONS			
ADD	RD, RS, RT	RD = RS + RT	(OVERFLOW TRAP)
ADDI	RD, RS, CONST16	RD = RS + CONST16±	(OVERFLOW TRAP)
ADDIU	RD, RS, CONST16	RD = RS + CONST16±	
ADDU	RD, RS, RT	RD = RS + RT	
CLO	RD, RS	RD = COUNTLEADINGONES(RS)	
CLZ	RD, RS	RD = COUNTLEADINGZEROS(RS)	
LA	RD, LABEL	RD = ADDRESS(LABEL)	
LI	RD, IMM32	RD = IMM32	
LUI	RD, CONST16	RD = CONST16 << 16	
MOVE	RD, RS	RD = RS	
NEGU	RD, RS	RD = –RS	
SEB[R2]	RD, RS	$RD = RS_{7:0}^{\pm}$	
SEH[R2]	RD, RS	$RD = RS_{15:0}^{\pm}$	
SUB	RD, RS, RT	RD = RS – RT	(OVERFLOW TRAP)
SUBU	RD, RS, RT	RD = RS – RT	

SHIFT AND ROTATE OPERATIONS		
ROTR[R2]	RD, RS, BITS5	$RD = RS_{BITS5-1:0} :: RS_{31:BITS5}$
ROTRV[R2]	RD, RS, RT	$RD = RS_{RT4:0-1:0} :: RS_{31:RT4:0}$
SLL	RD, RS, SHIFT5	RD = RS << SHIFT5
SLLV	RD, RS, RT	$RD = RS << RT_{4:0}$
SRA	RD, RS, SHIFT5	$RD = RS^{\pm} >> SHIFT5$
SRAV	RD, RS, RT	$RD = RS^{\pm} >> RT_{4:0}$
SRL	RD, RS, SHIFT5	$RD = RS^{\varnothing} >> SHIFT5$
SRLV	RD, RS, RT	$RD = RS^{\varnothing} >> RT_{4:0}$

LOGICAL AND BIT-FIELD OPERATIONS		
AND	RD, RS, RT	RD = RS & RT
ANDI	RD, RS, CONST16	$RD = RS \,\&\, CONST16^{\varnothing}$
EXT[R2]	RD, RS, P, S	$RS = RS_{P+S-1:P}^{\varnothing}$
INS[R2]	RD, RS, P, S	$RD_{P+S-1:P} = RS_{S-1:0}$
NOP		NO-OP
NOR	RD, RS, RT	RD = ~(RS | RT)
NOT	RD, RS	RD = ~RS
OR	RD, RS, RT	RD = RS | RT
ORI	RD, RS, CONST16	$RD = RS \mid CONST16^{\varnothing}$
WSBH[R2]	RD, RS	$RD = RS_{23:16} :: RS_{31:24} :: RS_{7:0} :: RS_{15:8}$
XOR	RD, RS, RT	RD = RS ⊕ RT
XORI	RD, RS, CONST16	$RD = RS \oplus CONST16^{\varnothing}$

CONDITION TESTING AND CONDITIONAL MOVE OPERATIONS		
MOVN	RD, RS, RT	IF RT ≠ 0, RD = RS
MOVZ	RD, RS, RT	IF RT = 0, RD = RS
SLT	RD, RS, RT	$RD = (RS^{\pm} < RT^{\pm})\ ?\ 1 : 0$
SLTI	RD, RS, CONST16	$RD = (RS^{\pm} < CONST16^{\pm})\ ?\ 1 : 0$
SLTIU	RD, RS, CONST16	$RD = (RS^{\varnothing} < CONST16^{\varnothing})\ ?\ 1 : 0$
SLTU	RD, RS, RT	$RD = (RS^{\varnothing} < RT^{\varnothing})\ ?\ 1 : 0$

MULTIPLY AND DIVIDE OPERATIONS		
DIV	RS, RT	$LO = RS^{\pm} / RT^{\pm};\ HI = RS^{\pm} \bmod RT^{\pm}$
DIVU	RS, RT	$LO = RS^{\varnothing} / RT^{\varnothing};\ HI = RS^{\varnothing} \bmod RT^{\varnothing}$
MADD	RS, RT	$ACC \mathrel{+}= RS^{\pm} \times RT^{\pm}$
MADDU	RS, RT	$ACC \mathrel{+}= RS^{\varnothing} \times RT^{\varnothing}$
MSUB	RS, RT	$ACC \mathrel{-}= RS^{\pm} \times RT^{\pm}$
MSUBU	RS, RT	$ACC \mathrel{-}= RS^{\varnothing} \times RT^{\varnothing}$
MUL	RD, RS, RT	$RD = RS^{\pm} \times RT^{\pm}$
MULT	RS, RT	$ACC = RS^{\pm} \times RT^{\pm}$
MULTU	RS, RT	$ACC = RS^{\varnothing} \times RT^{\varnothing}$

ACCUMULATOR ACCESS OPERATIONS		
MFHI	RD	RD = HI
MFLO	RD	RD = LO
MTHI	RS	HI = RS
MTLO	RS	LO = RS

JUMPS AND BRANCHES (NOTE: ONE DELAY SLOT)		
B	OFF18	$PC \mathrel{+}= OFF18^{\pm}$
BAL	OFF18	$RA = PC + 8,\ PC \mathrel{+}= OFF18^{\pm}$
BEQ	RS, RT, OFF18	$IF\ RS = RT,\ PC \mathrel{+}= OFF18^{\pm}$
BEQZ	RS, OFF18	$IF\ RS = 0,\ PC \mathrel{+}= OFF18^{\pm}$
BGEZ	RS, OFF18	$IF\ RS \geq 0,\ PC \mathrel{+}= OFF18^{\pm}$
BGEZAL	RS, OFF18	$RA = PC + 8;\ IF\ RS \geq 0,\ PC \mathrel{+}= OFF18^{\pm}$
BGTZ	RS, OFF18	$IF\ RS > 0,\ PC \mathrel{+}= OFF18^{\pm}$
BLEZ	RS, OFF18	$IF\ RS \leq 0,\ PC \mathrel{+}= OFF18^{\pm}$
BLTZ	RS, OFF18	$IF\ RS < 0,\ PC \mathrel{+}= OFF18^{\pm}$
BLTZAL	RS, OFF18	$RA = PC + 8;\ IF\ RS < 0,\ PC \mathrel{+}= OFF18^{\pm}$
BNE	RS, RT, OFF18	$IF\ RS \neq RT,\ PC \mathrel{+}= OFF18^{\pm}$
BNEZ	RS, OFF18	$IF\ RS \neq 0,\ PC \mathrel{+}= OFF18^{\pm}$
J	ADDR28	$PC = PC_{31:28} :: ADDR28^{\varnothing}$
JAL	ADDR28	$RA = PC + 8;\ PC = PC_{31:28} :: ADDR28^{\varnothing}$
JALR	RD, RS	RD = PC + 8; PC = RS
JR	RS	PC = RS

LOAD AND STORE OPERATIONS		
LB	RD, OFF16(RS)	$RD = MEM8(RS + OFF16^{\pm})^{\pm}$
LBU	RD, OFF16(RS)	$RD = MEM8(RS + OFF16^{\pm})^{\varnothing}$
LH	RD, OFF16(RS)	$RD = MEM16(RS + OFF16^{\pm})^{\pm}$
LHU	RD, OFF16(RS)	$RD = MEM16(RS + OFF16^{\pm})^{\varnothing}$
LW	RD, OFF16(RS)	$RD = MEM32(RS + OFF16^{\pm})$
LWL	RD, OFF16(RS)	RD = LOADWORDLEFT(RS + OFF16±)
LWR	RD, OFF16(RS)	RD = LOADWORDRIGHT(RS + OFF16±)
SB	RS, OFF16(RT)	$MEM8(RT + OFF16^{\pm}) = RS_{7:0}$
SH	RS, OFF16(RT)	$MEM16(RT + OFF16^{\pm}) = RS_{15:0}$
SW	RS, OFF16(RT)	$MEM32(RT + OFF16^{\pm}) = RS$
SWL	RS, OFF16(RT)	STOREWORDLEFT(RT + OFF16±, RS)
SWR	RS, OFF16(RT)	STOREWORDRIGHT(RT + OFF16±, RS)
ULW	RD, OFF16(RS)	RD = UNALIGNED_MEM32(RS + OFF16±)
USW	RS, OFF16(RT)	UNALIGNED_MEM32(RT + OFF16±) = RS

ATOMIC READ-MODIFY-WRITE OPERATIONS		
LL	RD, OFF16(RS)	RD = MEM32(RS + OFF16±); LINK
SC	RD, OFF16(RS)	IF ATOMIC, MEM32(RS + OFF16±) = RD; RD = ATOMIC ? 1 : 0

Registers		
0	zero	Always equal to zero
1	at	Assembler temporary; used by the assembler
2-3	v0-v1	Return value from a function call
4-7	a0-a3	First four parameters for a function call
8-15	t0-t7	Temporary variables; need not be preserved
16-23	s0-s7	Function variables; must be preserved
24-25	t8-t9	Two more temporary variables
26-27	k0-k1	Kernel use registers; may change unexpectedly
28	gp	Global pointer
29	sp	Stack pointer
30	fp/s8	Stack frame pointer or subroutine variable
31	ra	Return address of the last subroutine call

Default C Calling Convention (O32)

Stack Management
- The stack grows down.
  - Subtract from $sp to allocate local storage space.
  - Restore $sp by adding the same amount at function exit.
- The stack must be 8-byte aligned.
  - Modify $sp only in multiples of eight.

Function Parameters
- Every parameter smaller than 32 bits is promoted to 32 bits.
- First four parameters are passed in registers $a0–$a3.
  - 64-bit parameters are passed in register pairs:
    - Little-endian mode: $a1:$a0 or $a3:$a2.
    - Big-endian mode: $a0:$a1 or $a2:$a3.
- Every subsequent parameter is passed through the stack.
  - First 16 bytes on the stack are not used.
  - Assuming $sp was not modified at function entry:
    - The 1st stack parameter is located at 16($sp).
    - The 2nd stack parameter is located at 20($sp), etc.
  - 64-bit parameters are 8-byte aligned.

Return Values
- 32-bit and smaller values are returned in register $v0.
- 64-bit values are returned in registers $v0 and $v1:
  - Little-endian mode: $v1:$v0.
  - Big-endian mode: $v0:$v1.

MIPS32 Virtual Address Space				
kseg3	0xE000.0000	0xFFFF.FFFF	Mapped	Cached
ksseg	0xC000.0000	0xDFFF.FFFF	Mapped	Cached
kseg1	0xA000.0000	0xBFFF.FFFF	Unmapped	Uncached
kseg0	0x8000.0000	0x9FFF.FFFF	Unmapped	Cached
useg	0x0000.0000	0x7FFF.FFFF	Mapped	Cached

Reading The Cycle Count Register From C

```
unsigned mips_cycle_counter_read()
{
    unsigned cc;
    asm volatile("mfc0 %0, $9" : "=r" (cc));
    return (cc << 1);
}
```

Assembly-Language Function Example

```
# int asm_max(int a, int b)
# {
#   int r = (a < b) ? b : a;
#   return r;
# }

    .text
    .set    nomacro
    .set    noreorder

    .global asm_max
    .ent    asm_max

asm_max:
    move    $v0, $a0        # r = a
    slt     $t0, $a0, $a1   # a < b ?
    jr      $ra             # return
    movn    $v0, $a1, $t0   # if yes, r = b

    .end    asm_max
```

C / Assembly-Language Function Interface

```
#include <stdio.h>

int asm_max(int a, int b);

int main()
{
    int x = asm_max(10, 100);
    int y = asm_max(200, 20);
    printf("%d %d\n", x, y);
}
```

Invoking MULT And MADD Instructions From C

```
int dp(int a[], int b[], int n)
{
    int i;
    long long acc = (long long) a[0] * b[0];
    for (i = 1; i < n; i++)
        acc += (long long) a[i] * b[i];
    return (acc >> 31);
}
```

Atomic Read-Modify-Write Example

```
atomic_inc:
    ll      $t0, 0($a0)      # load linked
    addiu   $t1, $t0, 1      # increment
    sc      $t1, 0($a0)      # store cond'l
    beqz    $t1, atomic_inc  # loop if failed
    nop
```

Accessing Unaligned Data
note: ULW and USW automatically generate appropriate code

Little-Endian Mode		*Big-Endian Mode*	
LWR	RD, OFF16(RS)	LWL	RD, OFF16(RS)
LWL	RD, OFF16+3(RS)	LWR	RD, OFF16+3(RS)
SWR	RD, OFF16(RS)	SWL	RD, OFF16(RS)
SWL	RD, OFF16+3(RS)	SWR	RD, OFF16+3(RS)

Accessing Unaligned Data From C

```
typedef struct
{
    int u;
} __attribute__((packed)) unaligned;

int unaligned_load(void *ptr)
{
    unaligned *uptr = (unaligned *)ptr;
    return uptr->u;
}
```

MIPS SDE-GCC Compiler Defines	
__mips	MIPS ISA (= 32 for MIPS32)
__mips_isa_rev	MIPS ISA Revision (= 2 for MIPS32 R2)
__mips_dsp	DSP ASE extensions enabled
_MIPSEB	Big-endian target CPU
_MIPSEL	Little-endian target CPU
_MIPS_ARCH_*CPU*	Target CPU specified by `-march=`*CPU*
_MIPS_TUNE_*CPU*	Pipeline tuning selected by `-mtune=`*CPU*

Notes
- Many assembler pseudo-instructions and some rarely used machine instructions are omitted.
- The C calling convention is simplified. Additional rules apply when passing complex data structures as function parameters.
- The examples illustrate syntax used by GCC compilers.
- Most MIPS processors increment the cycle counter every other cycle. Please check your processor documentation.