Tentamen i kursen EDA330

Datorsystemteknik D

14/1 2000

Tentamensdatum: Fredag 14/1 2000 kl. 8.45 i sal MN

Examinator: Jonas Vasell

Institution: Datorteknik

Förfrågningar: Thomas Lundqvist (ankn. 1165)

Lösningar: Anslås måndag 17/1 på institutionens anslagstavla utanför laboratoriet

Resultat: Anslås senast fredag 28/1 på institutionens anslagstavla utanför laboratoriet

Rättningsgranskning: Fredag 28/1 kl. 11.30-12.00 hos Thomas Lundqvist, rum 6317, institutionen för datorteknik..

Betygsgränser: 3: 24-35 poäng, 4: 36-47 poäng, 5: 48-60 poäng

Tillåtna hjälpmedel: inga

Allmänt: För full poäng på en uppgift krävs både ett korrekt svar och en motivering. En bra motivering är minst lika viktig som ett korrekt svar. Redovisa noggrannt alla gjorda antaganden utöver de som anges i uppgiftstexten. Skriv tydligt och använd gärna figurer. Maximal poäng på varje deluppgift anges inom parentes efter uppgiftstexten.

Välgångsönskning: Lycka till!

Uppgifter (1-5):

1. I en minnesposition i en dator baserad på MIPS-processorn hittas vid ett visst tillfälle följande ord (på hexadecimal form): $\texttt{02801200}_{16}$.

   a. Om ordet tolkas som en instruktionskod, vilken är motsvarande instruktion, och vad gör den? En sammanställning över MIPS maskininstruktioner finns i bilaga 2. (2 p)

   b. Om ordet tolkas som ett flyttal på IEEE 754-format med enkel precision (8 bitars exponentrepresentation, och bias 127), vilket flyttal representerar det? Ge svaret på formen heltal $\times 2^{\text{exponent}}$, där heltal och exponent är decimaltal. (2 p)

   c. Om ordet tolkas som ett heltal i tvåkomplementsrepresentation, vilket heltal motsvarar det? Ge svaret på formen heltal $\times 2^{\text{exponent}}$, där heltal och exponent är decimaltal. (2 p)

2.

   a. Hur stort minnesutrymme räknat i bytes krävs som mest för att lagra sidtabeller i följande datorsystem med virtuellt minne? Sidstorleken är 16 KB, och det finns maximalt 64 processer med unika virtuella adressrum. Det virtuella adressrummet för varje process begränsas till byte-adresserna $0..2^{28}$-1. Fysiskt kan upp till 256 MB (1 MB = 1024 KB) adresseras. En approximation av LRU med 8 bitars tidstämplar tillämpas. Skrivningsstrategin är write back. För varje sida finns också fyra skyddsbitar som anger åtkomsträttigheter. Informationen om varje sidöversättning i sidtabellen måste lagras i ett helt antal bytes. (4 p)

   b. Förklara vad en TLB (Translation Lookaside Buffer) har för funktion i virtuell minneshantering. (2 p)

3. Vid arbete med implementering av en pipelinad MIPS-processor för en speciell klass av tillämpningar visar det sig att implementeringen av logik för att göra multiplikation i EX-steget är speciellt kritisk för prestanda och kostnad. I en lösning som vi här kallar baslösningen utförs alla multiplikationer på en klockcykel av speciell hårdvara i EX-steget. Om man kan låta multiplikationer ta två klockcykler istället för en, och därmed orsaka en stallcykel för varje multiplikation, så kan man i gengäld höja klockfrekvensen med 11%. Om man helt och hållet tar bort hårdvarustödet för multiplikation och istället utför alla multiplikationer med mjukvarurutiner så sänker man dessutom implementeringskostnaden med 8%. För att ta reda på om någon av dessa alternativa lösningar är vettig ur ett kostnads- och prestandaperspektiv så har man undersökt hur vanligt förekommande multiplikationer är i den aktuella klassen av tillämpningar. Det visade sig att endast 0,1% av alla exekverade instruktioner var multiplikationsinstruktioner. Din uppgift nu är att undersöka de olika implementeringsalternativen. (OBS! Även om denna uppgift ser omfattande ut, så är den konstruerad så att du kan hoppa över vissa deluppgifter om du skulle fastna.)

   a. Antag att CPI då baslösningen används är 1,4, samt att I/O-väntetiden för programmen är försumbar. Jämfört med baslösningen, hur mycket förändras då exekveringstiden för program genomsnittligt om man istället använder den lösning som tar två cykler? (2 p)

   b. Beskriv en metod för multiplikation av teckenlösa binära tal. Inkludera ett flödesschema i svaret. (2 p)

   c. Istället för att utföra multiplikationerna i hårdvara, så kan man som nämnts ovan utföra dem i mjukvara istället. Det innebär att man ersätter exekveringen av varje multiplikationsinstruktion med en sekvens av andra MIPS-instruktioner. Ett sätt att göra det är att låta assemblatorn behandla multiplikationsinstruktionerna som makroinstruktioner (även kallade syntetiserade instruktioner). Varje gång assemblatorn hittar en sådan instruktion ersätts den av en sekvens av andra instruktioner på ett sådant sätt att programmet före och efter multiplikationen inte påverkas. Det innebär att instruktionssekvensen inte får ändra innehållet i minnet, eller i andra register än de som modifieras av (i detta fall) multiplikationsinstruktionen eller som är reserverade för användning i makroinstruktioner. I MIPS assemblyspråk brukar registret `$at` vara reserverat för detta ändamål. Din uppgift är nu att med utgångspunkt i metoden från deluppgift b skriva en sådan instruktionssekvens för instruktionen `multu $r1, $r2`, där `$r1` och `$r2` representerar godtyckliga register (förutom de som är reserverade för assemblatorn). Du får använda de instruktioner som listas i bilaga 2 (förutom multiplikations- och divisionsinstuktionerna) plus instruktionerna `mthi` och `mtlo` som motsvarar `mfhi` respektive `mflo` men flyttar värden i motsatt riktning. Dessutom får du lov att anta att registren `$t8` och `$t9` liksom `$at` är reserverade för assemblatorn och alltså fritt kan användas som temporära register i din rutin, samt att inget operandvärde kan vara större än 80000000_{16}. (8 p)

   d. Antag att med lösningen i deluppgift c så är CPI för all kod inklusive multiplikationssekvenserna densamma som i baslösningen. Vad händer då med prestanda räknat på motsvarande sätt som i deluppgift a om man använder lösningen från deluppgift c? Jämför prestandaförändringen med kostnadsförändringen. Om du inte löst deluppgift c, så beskriv hur du skulle räkna ut svaret på denna uppgift. (2 p)

e. Lösningen i deluppgift c har den nackdelen att programmen måste assembleras speciellt för den aktuella MIPS-implementeringen. I praktiken ställer det till problem eftersom det visar sig att man i många fall endast har objektkod (redan assemblerade och länkade program) att tillgå. En vanlig lösning i sådana fall är att låta de oimplementerade instruktionerna generera en exception (mjukvaruavbrott, kallas ofta också för trap). I MIPS finns redan en speciell instruktion, `break`, som genererar denna typ av avbrott (den saknas dock i bilaga 2). Vi antar nu att denna lösning tillämpas i detta fall. Varje gång en oimplementerad instruktion (som t.ex. en multiplikationsinstruktion) når fram till EX-steget så genereras ett avbrott med orsakskod 10. Orsakskoden lagras i bitarna 2-6 i Cause-registret (se bilaga 2), och adressen till efterföljande instruktion lagras i EPC-registret, medan alla instruktioner i pipelinen fram till och med EX kastas bort. Dessutom införs utöver Cause och EPC två nya register i coprocessor 0 som kallas Oper1 och Oper2, där operandvärdena till den instruktion som orsakade avbrottet lagras. Din uppgift är nu att skriva en avbrottshanterare för oimplementerade instruktioner som kontrollerar om en multiplikationsinstruktion (`multu $r1, $r2`) orsakat avbrottet, och i så fall utför multiplikationen så att det vid återhopp efter avbrottet ser ut precis som om multiplikationsinstruktionen hade utförts i hårdvara. Antag att koden från deluppgift c finns tillgänglig i form av en subrutin `MULTU` som tar sina inparametrar från registren `$a0` och `$a1`. (6 p)

4. Följande program körs på en MIPS-processor utan pipelining (nästa instruktion hämtas inte förrän föregående instruktion är helt avslutad, varför det inte förekommer någon delayed branch):

```
200: lw   $3, 0($2)
204: lw   $4, 100($2)
208: slti $6, $2, 16
212: add  $5, $3, $4
216: sw   $5, 100($2)
220: addi $2, $2, 4
224: bne  $6, $0, -28
228: add $0, $0, $0
```

Varje programrad ovan inleds med byte-adressen till det ord där instruktionen ligger lagrad. Alla konstanter, inklusive adresser, är decimala (bas 10).

Processorn ingår i ett datorsystem med ett gemensamt cacheminne för instruktioner och data. Cacheminnet är tvåvägs partiellt associativt (two way set associative) med blockstorleken 2 ord och total datalagringskapacitet 16 ord. Utbytesalgoritmen är LRU. Vid skrivmissar hämtas det aktuella blocket alltid in till cacheminnet.

a. Lista den sekvens av minnesreferenser i form av byte-adresser som programmet ger upphov till. Registret $2 innehåller värdet noll innan programkörningen. (2 p)

b. Beskriv hur tag, index, och block-offset beräknas utgående från en byte-adress i detta fall. (2 p)

c. Ange för var och en av de första 10 minnesreferenserna om den ger träff eller miss i cacheminnet. Ange också i tabellform innehållet i varje cacheblock efter varje referens genom att ange tag för det minnesblock som ligger där. Utgå ifrån att cacheminnet är tomt från början. Vad blir hit rate för de första 10 minnesreferenserna? (4 p)

d. Fortsätt som i deluppgift c för de nästkommande 20 minnesreferenserna. Starta med cacheinnehållet som det såg ut efter tionde referensen. Vad blir hit rate för de första 30 minnesreferenserna? (2 p)

e. Vad skulle ändras om programmet istället kördes på en MIPS-processor med pipeline liknande den i bilaga 1, med en cykels delayed branch och hoppberäkning i ID-steget? Skulle det rent principiellt kunna påverka hit rate? (2 p)

f. Antag att vi använder en pipelinad processor som den i deluppgift e, att miss penalty vid miss i cacheminnet är 5 cykler, och att cacheminnet enbart kan hantera en minnesåtkomst i taget. En ny minnesåtkomst kan alltså inte påbörjas förrän föregående åtkomst är avklarad. Om en dataåtkomst och en instruktionsåtkomst försöker starta samtidigt, så ges prioritet till dataåtkomsten. Antag också att cacheminnet är av write-through-typ och har en skrivbuffer så att skrivningar alltid kan avslutas på en cykel om data finns i cacheminnet, och utskrivning av block till primärminnet inte orsakar någon väntetid. Ange den sekvens av minnesreferenser i form av byte-adresser som genereras. Det räcker med de 10 första. (4 p)

5. Nedan följer ett antal frågor med tre svarsalternativ (1, X, 2) vardera, varav endast ett är rätt. Ställ upp svaren som en tipsrad. Varje rätt svar ger en poäng. (12 p)

   a. Om ett cacheminne innehåller ett set, vilken typ av cacheminne är det då? (1) Fullt associativt. (X) Partiellt associativt. (2) Icke-associativt.

   b. För vilken typ av bussar är följande utmärkande egenskaper; standardiserad, variabel bandbredd, lång? (1) Bakplansbuss. (X) I/O-buss. (2) Processor-minnebuss.

   c. Vad är det huvudsakliga problemet med en synkron buss? (1) Långsam. (X) Klockdistribution. (2) Kräver handskakning.

   d. Vad innebär multiplexing? (1) Samma ledare används för samma signal vid olika tidpunkter. (X) Separata ledare används för samma signal vid olika tidpunkter. (2) Samma ledare används för olika signaler vid olika tidpunkter.

   e. Vilka faktorer inverkar på prestandamåttet MIPS? (1) Antal exekverade instruktioner och CPI. (X) Instruktionsuppsättning och klockfrekvens. (2) Antal exekverade instruktioner och tillämpningsprogram.

   f. Ungefär hur mycket ökar prestanda för datorsystem per år? (1) Cirka 30%. (X) Cirka 50%. (2) Cirka 80%.

   g. Hur utvecklas skillnaden i snabbhet mellan minnes- och processorkomponenter? (1) Den ökar. (X) Den är i stort sett konstant. (2) Den minskar.

   h. Vad är ett exakt avbrott? (1) Ett avbrott vars ursprung är helt känt. (X) Ett avbrott från vilket återhopp aldrig sker. (2) Ett avbrott som hanteras omedelbart.

   i. Vilken typ av pipelinekonflikter kan undvikas genom data forwarding? (1) Datakonflikter. (X) Styrkonflikter. (2) Resurskonflikter.

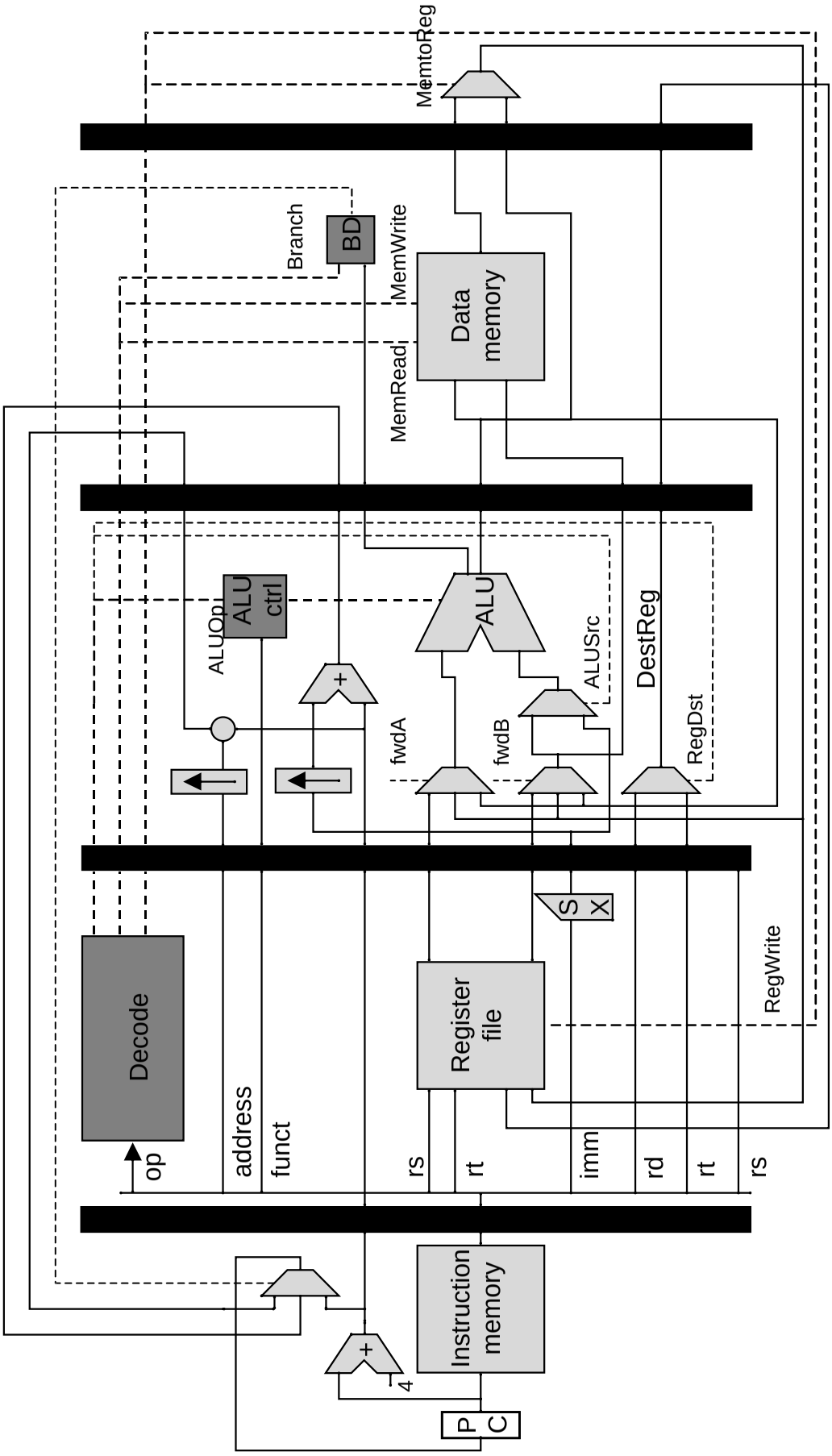
   j. Vad handlar koherens i datorsystem om? (1) En processor ska bara kunna se den senast lagrade versionen av data på varje adress. (X) De protokoll som används ska vara lättbegripliga. (2) Man kan inte använda cacheminnen om det finns mer än en processor.

   k. Vad innebär tidslokalitet? (1) Om en minnesadress refereras så refereras ofta en närliggande adress snart. (X) Om en minnesadress refereras så refereras den snart igen. (2) Om en minnesadress refereras så refereras ofta nästföljande adress snart.

   l. Vilka egenskaper i en minnehierarki påverkas genom att variera blockstorlek? (1) Utnyttjande av rumslokalitet och överföringstid. (X) Utnyttjande av både tids- och rumslokalitet. (2) Utnyttjande av tidslokalitet och överföringstid.

SLUT

Bilaga 1: MIPS pipeline



Bilaga 2: MIPS maskininstruktioner

Notes:*op*, *funct*, *rd*, *rs*, *rt*, *imm*, *address*, *shamt* refer to fields in the instruction format. The program counter PC is assumed to point to the next instruction (usually 4 + the address of the current instruction). M is the byte-addressed main memory.

Assembly instruction	Instr. format	*op* *op*/*funct*	Meaning	Comments
`add $rd, $rs, $rt`	R	0/32	`$rd = $rs + $rt`	Add contents of two registers
`sub $rd, $rs, $rt`	R	0/34	`$rd = $rs - $rt`	Subtract contents of two registers
`addi $rt, $rs, imm`	I	8	`$rt = $rs + imm`	Add signed constant
`addu $rd, $rs, $rt`	R	0/33	`$rd = $rs + $rt`	Unsigned, no overflow
`subu $rd, $rs, $rt`	R	0/35	`$rd = $rs - $rt`	Unsigned, no overflow
`addiu $rt, $rs, imm`	I	9	`$rt = $rs + imm`	Unsigned, no overflow
`mfc0 $rt, $rd`	R	16	`$rt = $rd`	*rd* = coprocessor register (e.g. epc, cause, status)
`mult $rs, $rt`	R	0/24	`Hi, Lo = $rs * $rt`	64 bit signed product in Hi and Lo
`multu $rs, $rt`	R	0/25	`Hi, Lo = $rs * $rt`	64 bit unsigned product in Hi and Lo
`div $rs, $rt`	R	0/26	`Lo = $rs / $rt, Hi = $rs mod $rt`	
`divu $rs, $rt`	R	0/27	`Lo = $rs / $rt, Hi = $rs mod $rt` (unsigned)	
`mfhi $rd`	R	0/16	`$rd = Hi`	Get value of Hi
`mflo $rd`	R	0/18	`$rd = Lo`	Get value of Lo
`and $rd, $rs, $rt`	R	0/36	`$rd = $rs & $rt`	Logical AND
`or $rd, $rs, $rt`	R	0/37	`$rd = $rs | $rt`	Logical OR
`andi $rt, $rs, imm`	I	12	`$rt = $rs & imm`	Logical AND, unsigned constant
`ori $rt, $rs, imm`	I	13	`$rt = $rs | imm`	Logical OR, unsigned constant
`sll $rd, $rs, shamt`	R	0/0	`$rd = $rs << shamt`	Shift left logical (shift in zeros)
`srl $rd, $rs, shamt`	R	0/2	`$rd = $rs >> shamt`	Shift right logical (shift in zeros)
`lw $rt, imm($rs)`	I	35	`$rt = M[$rs + imm]`	Load word from memory
`sw $rt, imm($rs)`	I	43	`M[$rs + imm] = $rt`	Store word in memory
`lbu $rt, imm($rs)`	I	37	`$rt = M[$rs + imm]`	Load a single byte, set bits 8-31 of $*rt* to zero
`sb $rt, imm($rs)`	I	41	`M[$rs + imm] = $rt`	Store byte (bits 0-7 of $*rt*) in memory
`lui $rt, imm`	I	15	`$rt = imm * 2^16`	Load constant in bits 16-31 of register $*rt*
`beq $rs, $rt, imm`	I	4	`if($rs==$rt) PC = PC + imm` (PC always points to next instruction)	
`bne $rs, $rt, imm`	I	5	`if($rs!=$rt) PC = PC + imm` (PC always points to next instruction)	
`slt $rd, $rs, $rt`	R	0/42	`if($rs<$rt) $rd = 1; else $rd = 0`	
`slti $rt, $rs, imm`	I	10	`if($rs<imm) $rt = 1; else $rt = 0`	
`sltu $rd, $rs, $rt`	R	0/43	`if($rs<$rt) $rd = 1; else $rd = 0` (unsigned numbers)	
`sltiu $rt, $rs, imm`	I	11	`if($rs<imm) $rt = 1; else $rt = 0` (unsigned numbers)	
`j destination`	J	2	`PC = address*4`	Jump to *destination*, *address* = *destination*/4
`jal destination`	J	3	`$ra = PC; PC = address*4` (Jump and link, *address* = *destination*/4)	
`jr $rs`	R	0/8	`PC = $rs`	Jump to address stored in register $*rs*

MIPS registers

Name	Number	Usage
`$zero`	0	constant 0
`$at`	1	reserved for assembler
`$v0 - $v1`	2-3	expression evaluation and function results
`$a0 - $a3`	4-7	arguments
`$t0 - $t7`	8-15	temporary, saved by caller
`$s0 - $s7`	16-23	temporary, saved by called function
`$t8 - $t9`	24-25	temporary, saved by caller
`$k0 - $k1`	26-27	reserved for kernel (OS)
`$gp`	28	points to middle of a 64K block in the data segment
`$sp`	29	stack pointer (top of stack)
`$fp`	30	frame pointer (beginning of current frame)
`$ra`	31	return address
`Hi, Lo`	-	store partial result of mult and div operations
`PC`	-	contains the address of the next instruction to be fetched (this is not a real MIPS register, and is only used to define instructions)
`status`	-	register 12 in coprocessor 0, stores interrupt mask and enable bits
`cause`	-	register 13 in coprocessor 0, stores exception type and pending interrupt bits
`epc`	-	register 14 in coprocessor 0, stores address of instruction causing exception

MIPS Instruction formats

Format	Bits 31-26	Bits 25-21	Bits 20-16	Bits 15-11	Bits 10-6	Bits 5-0
R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	imm		
J	op	address				

MIPS Assembler syntax

```
                              # This is a comment
          .data               # Store following data in the data
                              # segment
items:                        # This is a label connected to the
                              # next address in the current segment
          .word 1, 2          # Stores values 1 and 2 in next two
                              # words
hello:    .asciiz “Hello”     # Stores null-terminated string in
                              # memory
          .text               # Store following instructions in
                              # the text segment
main:     lw $t0, items($zero)  # Instruction that uses a label to
                              # address data
```