

1. **NOTA:** Use no máximo 40 palavras para responder a cada uma das 4 questões seguintes:

- Apresente, justificando, a decomposição em instruções nativas da instrução “**bge \$4, \$5, else**”.
- Apresente o formato de codificação da instrução “**J**”, referindo quais os campos em que é decomposta a dimensão de cada um e o seu significado. Explique ainda como essa instrução é executada.
- Indique o modo de codificação do expoente na norma IEEE-754, precisão simples, e apresente a razão pela qual se utiliza esse método.
- Apresente a razão pela qual, num datapath *pipelined*, as instruções do tipo R passam pela fase “memory access”.

2. Considere o *datapath single-cycle* que foi apresentado nas aulas teóricas. Admita os seguintes atrasos de propagação envolvidos nos vários elementos operativos e de estado:

Memória externa (dados e código):      Leitura – 3ns;   Escrita – 9ns

File Register: Leitura – 3ns;   Escrita – 4ns                      Sign Extend: 4ns                      Shifter: 3ns

ALU (qualquer operação): 5ns    Somadores: 5ns                      Outros dispositivos: 0ns

- Determine o tempo mínimo necessário à execução de cada uma das três instruções seguintes (consideradas independentemente) e calcule a máxima frequência de relógio desta arquitectura. Justifique a sua resposta.

**sw    \$a0, 0(\$s0) [45%]                      add   \$t1, \$t2, \$a0 [35%]                      beq   \$s12, \$0, loop [20%]**

- Considerando apenas as três instruções da alínea anterior, que a frequência de relógio é de 25MHz, e que a taxa média de ocorrência é a indicada entre parêntesis rectos, determine o valor médio do tempo durante o qual o *datapath* não sofre alterações em cada ciclo.

3. Considere que o conteúdo dos registos **\$f6** e **\$f4** é, respectivamente:

**\$f6 = 0.90625<sub>10</sub> × 2<sup>-58</sup>                      \$f4 = -16.3125<sub>10</sub> × 2<sup>65</sup>**

- Obtenha a representação em binário das quantidades armazenadas naqueles registos (codificadas segundo a norma IEEE 754, precisão simples).
- Determine o resultado da instrução **div.s \$f2, \$f4, \$f6**, realizando, em binário e/ou hexadecimal, os passos necessários. Indique, em binário (de acordo com a norma IEEE 754, precisão simples), qual o conteúdo do registo **\$f2** após a execução da instrução.

4. Considere o *datapath* e a unidade de controlo fornecido na Figura 1 (ligeiramente alterado em relação à versão das aulas teóricas), sabendo que corresponde a uma implementação multi-ciclo simplificada do MIPS, sem *pipelining*.

- Preencha a tabela fornecida em anexo com o nome de cada uma das fases de execução da instrução “**sub \$16, \$15, \$11**” e com o valor que tomam, em cada uma delas, os sinais e valores do *datapath* e os sinais de controlo ali indicados. Admita que o valor lógico “1” corresponde ao estado activo (considere que os registos, antes da execução da instrução, têm os seguintes valores: **\$15=0xA0F4, \$11=0x100F4F0A, \$PC=0x4000C0**).
- Admita que os valores indicados no *datapath* fornecido correspondem à “fotografia” tirada no decurso da execução de uma dada instrução. Identifique, observando com atenção todos os sinais: **1)** qual a instrução em causa (apresente a instrução completa); **2)** qual a fase de execução em que se encontra; **3)** qual o valor do PC após a execução da instrução. Justifique todos os passos da sua resposta.

5. Considere, na Figura 2, as tabelas ali apresentadas. Admita que o valor presente em **\$PC** corresponde ao endereço da primeira instrução, que nesse instante o conteúdo dos registos é o indicado, e que vai iniciar-se o “*instruction fetch*” dessa instrução. Considere ainda o *datapath* e a unidade de controlo fornecido na Figura 1.

- Escreva, em *Assembly* do MIPS, o trecho de código correspondente às seis instruções presentes na tabela da direita e indique, para cada uma delas, o endereço de memória em que se encontra. Crie *labels* sempre que tal seja adequado. Justifique todos os passos da sua resposta.
- Face aos valores presentes no segmento de dados (tabela da esquerda) e ao código que obteve na alínea anterior, determine, justificando, o número total de ciclos de relógio que demora a execução completa desse trecho de código (desde o instante inicial do *instruction fetch* da primeira instrução até ao momento em que vai iniciar-se o *instruction fetch* da instrução presente em “**next:**”), bem como o valor final do registo **\$6**.
- Represente, sob a forma de um diagrama temporal, a evolução do sinal de relógio e dos sinais de controlo “**PCWrite**”, “**RegWrite**”, “**RegDst**”, “**IRWrite**” e “**ALUOp**” durante a execução (em sequência e pela ordem apresentada) das 3 primeiras instruções do trecho de código apresentado na Figura 2 (Nota: represente “don’t care” por **////**). Justifique adequadamente a sua resposta.

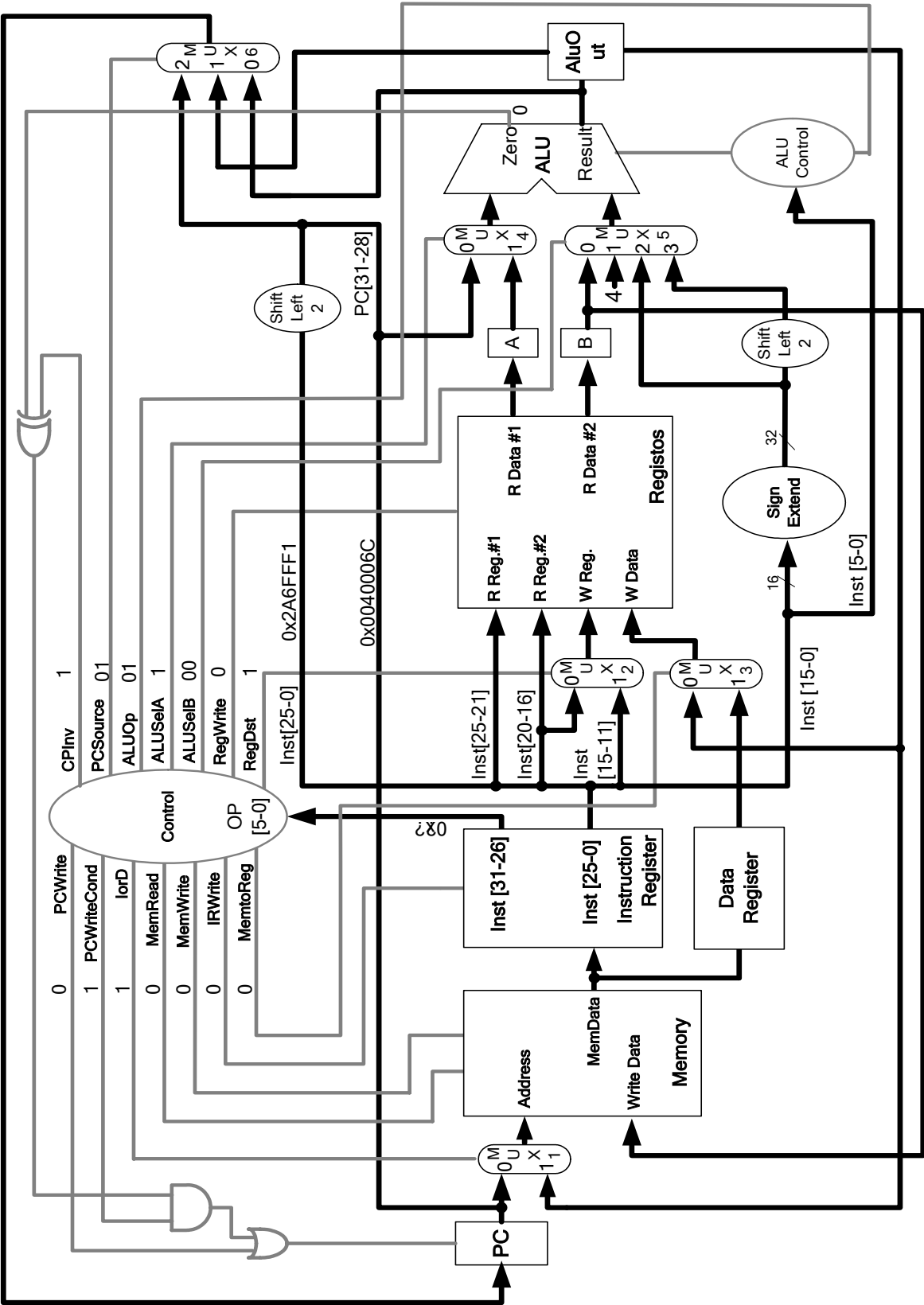


Figura 1



universidade de aveiro

Departamento de Electrónica e Telecomunicações  
Arquitectura de Computadores I  
Exame Final (Resolução) – 03.01.2006

1.

- a) As instruções nativas utilizadas para saltos condicionais são beq, bne e slt. O salto é efectuado de  $\$4 \geq \$5$ . Mas  $\$4 \geq \$5 \Leftrightarrow \sim(\$4 < \$5)$ . Decompondo em instruções nativas tem-se

slt     \$1, \$4, \$5  
beq     \$1, \$0, else

b)

opcode (6 bits)	address (26 bits)
--------------------	----------------------

O campo address é multiplicado por 4 (o resultado da multiplicação tem 28 bits) e é de seguida concatenado aos 4 bits mais significativos do PC (Program Counter).

c)

S (1 bit)	E (8 bits)	F (23 bits)
--------------	---------------	----------------

A representação é feita em sinal e módulo.

S representa o sinal (0 para positivos e 1 para negativos)

E é o expoente codificado em excesso 127 (Expoente real =  $E - 127$ )

F é a mantissa

Se este método não fosse utilizado, não seria possível saber quantos dígitos reservar para a parte inteira e para a parte fraccionária, sabendo que o espaço de armazenamento é limitado.

- d) O datapath pipelined está dividido em 5 fases. Como a fase “Memory Access” (MEM) aparece antes da fase de “Write Back” (WB), as instruções do tipo R têm que passar por essa fase para poderem chegar à fase WB (fase necessária à sua conclusão). Na figura seguinte mostra-se um problema que poderia ocorrer caso as instruções do tipo R não passassem pela fase MEM. Trata-se de uma instrução de load seguida de uma instrução do tipo R. Do lado esquerdo mostra-se o que é correcto (todas as instruções passam pelas 5 fases) e no lado direito mostra-se o problema. As duas instruções estão a tentar escrever no banco de registos ao mesmo tempo.

IF	ID	EX	MEM	WB	
	IF	ID	EX	MEM	WB

IF	ID	EX	MEM	WB
	IF	ID	EX	WB

2.

a) SW

$$T = \text{Somador} + (\text{Sign Extend} + \text{Shifter} + \text{Somador}) + \text{ALU} + \text{Escrever memória} = \\ = 5 + (4 + 3 + 5) + 5 + 9 = 31 \text{ ns}$$

TIPO-R

$$T = \text{Somador} + \text{Ler File Register} + \text{ALU} + \text{Escrever File Register} = \\ = 5 + 3 + 5 + 4 = 17 \text{ ns}$$

Branch

$$T = \text{Somador} + (\text{Sign Extend} + \text{Shifter} + \text{Somador}) + \text{ALU} = \\ = 5 + (4 + 3 + 5) + 5 = 22 \text{ ns}$$

Considerando o set de instruções apenas com estas três instruções, a máxima frequência do relógio é o inverso do maior atraso, ou seja

$$f = 1/31\text{ns} = 32 \text{ MHz}$$

b)  $f = 25 \text{ MHz} \Rightarrow T = 1/25 \mu\text{s} = 0.04 \mu\text{s} = 40 \text{ ns}$  (tempo para cada instrução)

$$T' = 0.45 \cdot 31 + 0.35 \cdot 17 + 0.20 \cdot 22 = 24.3 \text{ ns}$$

$T' / T = 61\%$  (aprox.) é a utilização do datapath.

3.

a)  $0.90625$  (decimal)  $= 0.11101 = 1.1101 \cdot 2^{-1}$  (binário)  
 $-16.3125$  (decimal)  $= -10000.0101 = -1.00000101 \cdot 2^4$

Os valores a codificar são  $1.1101 \cdot 2^{-59}$  e  $-1.00000101 \cdot 2^{69}$

1º valor:

$$S = 0$$

$$E = -59 + 127 = 68 \text{ (decimal)} = 01000100 \text{ (binário)}$$

$$F = 110100000000000000000000$$

2º valor

$$S = 1$$

$$E = 69 + 127 = 196 \text{ (decimal)} = 11000100 \text{ (binário)}$$

$$F = 000001010000000000000000$$

Os valores normalizados para IEEE-754 são portanto

$$0 \ 01000100 \ 110100000000000000000000 \Leftrightarrow 0x22680000$$

$$1 \ 11000100 \ 000001010000000000000000 \Leftrightarrow 0xE2028000$$

- b) Expoente do resultado:  $69 - (-59) = 128$   
 Sinal do resultado :  $1 \text{ xor } 0 = 1$  (o resultado é negativo)

```

1.00000101 | 1.1101
   111      .1001
   1110
   11101
     0

```

O resultado é:  $-0.1001 * 2^{128} = -1.001 * 2^{127}$

No formato IEEE-754 fica:

1 11111110 000100000000000000000000  $\Leftrightarrow$  0xFF100000

4. A tabela vem preenchida na última página.

- a) sub \$16, \$15, \$11

Código máquina

```

000000 01111 01011 10000 00000 100010
opcode  rs    rt    rd    shamt funct

```

Branch Target

$$0x8022 * 4 + PC = 0x20088 + 0x4000C8 = 0x0042014C$$

Operação \$15 - \$11

$$0x000A0F4 - 0x100F4F0A = 0xEFF151EA$$

- b)

- 1)

PCWrite=0  
 PCWriteCond=1

Conclui-se que é uma instrução do tipo branch

ALUOp=01 (subtração)  
 CPIInv=1 (se a ALU devolver 0 o salto é efectuado – beq)

Trata-se da instrução beq (opcode = 0x04)

Inst[25-0] = 0x2A6FFF1

000100 10101 00110 1111111111110001

O offset está representado em complemento para 2, o seu valor é -0xF.  
Para calcular o branch target address efectua-se a seguinte operação

$(PC+4) + 4*offset \rightarrow 0x0040006C - 0x3C = 0x00400030$

A instrução é finalmente,  
beq \$21, \$6, 0x00400030

2) Está na frase Branch Conclusion (3ª e última fase) já que PCWriteCond=1 e ALUZero = 0 (a ALU já fez a subtracção e já tem o resultado disponível).

3) Como o salto não é efectuado (PCWriteCond=1 e CPIInv=1 e ALUZero=0) então o valor do PC após a execução da instrução será o Branch Target, ou seja,

0x00400030

5.

a)

100011 00111 00101 0000000000000100  $\Leftrightarrow$  lw \$5, 4(\$7)  
lw

000000 00101 01001 00101 00000 100100  $\Leftrightarrow$  and \$5, \$5, \$9  
tipoR and

000100 00101 01000 0000000000000011  $\Leftrightarrow$  beq \$5, \$8, next  
beq offset=3  
 $(PC+4)+4*offset = 0x00400038 + 0xC = 0x00400044$  (label next)

000000 00111 01010 00111 00000 100000  $\Leftrightarrow$  add \$7, \$7, \$10  
tipoR add

001000 00110 00110 1111111111111111  $\Leftrightarrow$  addi \$6, \$6, -1  
addi offset = -1

000101 00110 00000 1111111111111010  
bne offset = -6  
 $(PC+4)+4*offset = 0x00400044 - 0x18 = 0x0040002C$  (label start)

Endereço	Código
0x0045002C	start: lw \$5, 4(\$7)
0x00400030	and \$5, \$5, \$9
0x00400034	beq \$5, \$8, next
0x00400038	add \$7, \$7, \$10
0x0040003C	addi \$6, \$6, -1
0x00400040	bne \$6, \$0, start
0x00400044	next: ...

b)

\$5 = 0x0021B5C3  
 \$5 = 0x000000C3  
 \$5 != \$8 → o salto não é efectuado  
 \$7 = 0x10010098  
 \$6 = 0x0000001B  
 \$6 != 0 → o salto é efectuado para “start”

\$5 = 0x0021B54E  
 \$5 = 0x0000004E  
 \$5 != \$8 → o salto não é efectuado  
 \$7 = 0x10010094  
 \$6 = 0x0000001A  
 \$6 != 0 → o salto é efectuado para “start”

\$5 = 0x0021B52F  
 \$5 = 0x0000002F  
 \$5 != \$8 → o salto não é efectuado  
 \$7 = 0x10010090  
 \$6 = 0x00000019  
 \$6 != 0 → o salto é efectuado para “start”

\$5 = 0x0021B581  
 \$5 = 0x00000081  
 \$5 == \$8 → o salto é efectuado para “next”

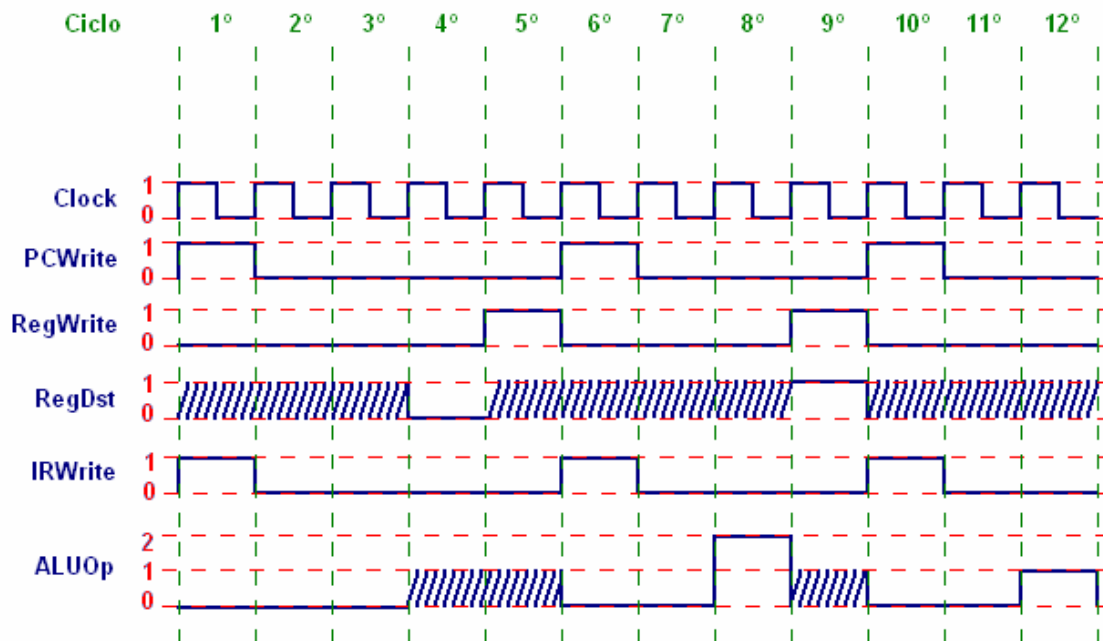
Note-se que:

Instrução	Nº de ciclos necessários	Nº de vezes que é utilizada
lw	5	4
and	4	4
beq	3	4
add	4	3
addi	4	3
bne	3	3

O nº de ciclos total é, portanto:  $5*4 + 4*4 + 3*4 + 4*3 + 4*3 + 3*3 = 81$  ciclos

O valor final em \$6 é 0x00000019

c) Para executar as 3 instruções são necessários no total  $5+4+3 = 12$  ciclos.




- O PCWrite só toma o valor 1 no 1º ciclo de cada instrução (Instruction Fetch)
- O RegWrite é 1 no último ciclo do lw e no último ciclo do and para que se possa escrever no registo destino. O branch não escreve no banco de registos.
- O RegDst pode tomar qualquer valor (don't care) quando o RegWrite é 0. Quando este é 1, o RegDst toma o valor 0 para o lw (escreve no registo rt) e toma o valor 1 para o and (escreve no registo rd).
- O IRWrite apenas toma o valor 1 durante o Instruction Fetch.
- O ALUOp é sempre 00 (soma) no 1º e no 2º ciclos para poder calcular o PC+4 e o branch target, respectivamente. No 3º ciclo é 00 para o lw para calcular rs+offset (endereço de memória a aceder) e para o and é 10 para que a operação da ALU dependa do campo funct (instruções tipo R). No caso do branch é 01 (subtracção) para efectuar a subtracção que permite comparar os registos rs e rt através da saída "Zero" da ALU.
- A partir do 3º ciclo, o ALUOp é "don't care" porque a ALU deixa de ser necessária.



### Anexo: Tabela do Exercício 4

Fase 1	Fase 2	Fase 3	Fase 4	Fase 5
--------	--------	--------	--------	--------

Nome da fase	Instruction Fetch	Instruction Decode	Operation Execute	Write Back	
--------------	-------------------	--------------------	-------------------	------------	---

<i>Datapath</i>					
PC	0x004000C0	0x004000C4	0x004000C4	0x004000C4	
Instr. Register	?	0x01EB8022	0x01EB8022	0x01EB8022	
Data Register	?	0x01EB8022	?	?	
A	?	?	0x0000A0F4	0x0000A0F4	
B	?	?	0x100F4F0A	0x100F4F0A	
ALU Result	0x004000C4	0x0042014C	0x1FF151EA	?	
ALU Out	?	0x004000C4	0x0042014C	0x1FF151EA	
ALU Zero	0	0	0	?	

Controlo					
PCSource	00	XX	XX	XX	
lorD	0	X	X	X	
PCWriteCond	X	0	0	0	
PCWrite	1	0	0	0	
RegWrite	0	0	0	1	
RegDst	X	X	X	1	
MemWrite	0	0	0	0	
MemtoReg	X	X	X	0	
MemRead	1	0	0	0	
IRWrite	1	0	0	0	
ALUOp	00	00	10	XX	
ALUSelB	01	11	00	XX	
ALUSelA	0	0	1	X	

Endereço ...	Dados ...	OpCode	Funcnt	Operação	CPU		Endereço	Código
0x10010090	0x0021B500	0	0x20	add	reg \$5	0x0045007C	0x0040002C	0x8CE50004
		0	0x22	sub	reg \$6	0x0000001C	0x00400030	0x00A92824
0x10010094	0x0021B581	0	0x24	and	reg \$7	0x1001009C	0x00400034	0x10A80003
		0	0x25	or	reg \$8	0x00000081	0x00400038	0x00EA3820
0x10010098	0x0021B52F	0x02		j	reg \$9	0x000000FF	0x0040003C	0x20C6FFFF
		0x04		beq	reg \$10	0xFFFFFFF7	0x00400040	0x14C0FFFA
0x1001009C	0x0021B54E	0x05		bne	\$PC	0x0040002C	next:	...
		0x08		addi	CPU			
0x100100A0	0x0021B5C3	0x0C		andi				
		0x23		lw				
0x100100A4	0x0021B5FF	0x2b		sw				
...	...							