Universidade de Brasília
Departamento de Ciência da Computação

Aula 4
Arquitetura MIPS

Arquiteturas de Processadores

Principais arquiteturas hoje

ARMv7 (32 bits) e ARMv8 (64 bits)

x86 (32 bits) e x64 (64 bits)

Vamos usar a arquitetura do processador MIPS32 como exemplo de aplicação dos conceitos básicos em um projeto completo.

http://www.mips.com *http://www.imgtec.com*

MIPS=**M**icroprocessor without **I**nterlocked **P**ipeline **S**tages

Arquitetura projetada na universidade de Stanford em 1981.
Empresa fundada em 1984.
A MIPS Inc. foi comprada pela Imagination Tech. em Fev de 2013.

O MIPS:

- Semelhante a outras arquiteturas RISC desenvolvidas desde a década de 1980 (ARM, SunSparc,...)
- Mais de 400 milhões de processadores MIPS fabricados por ano
- Foi usada pela NEC, Nintendo, Cisco, Broadcom, Silicon Graphics, Sony...
- MIPS 32 e 64 bits, e coprocessadores (3D,DSP,etc)!
- Preparado para Android e Linux.

The end of x86: first MIPS netbook appears

AzmountAryl
K7 Athlon XP (Barton) Senior Boarder

Posts: 974
Joined: Sat Mar 03, 2007 8:49 am

by **AzmountAryl** » Sat Apr 11, 2009 11:30 am

Who could think that it is even possible to contest the mighty IntEl and their ugly x86 arch in the cheap end of computer market?
Well Chinese have open minds.

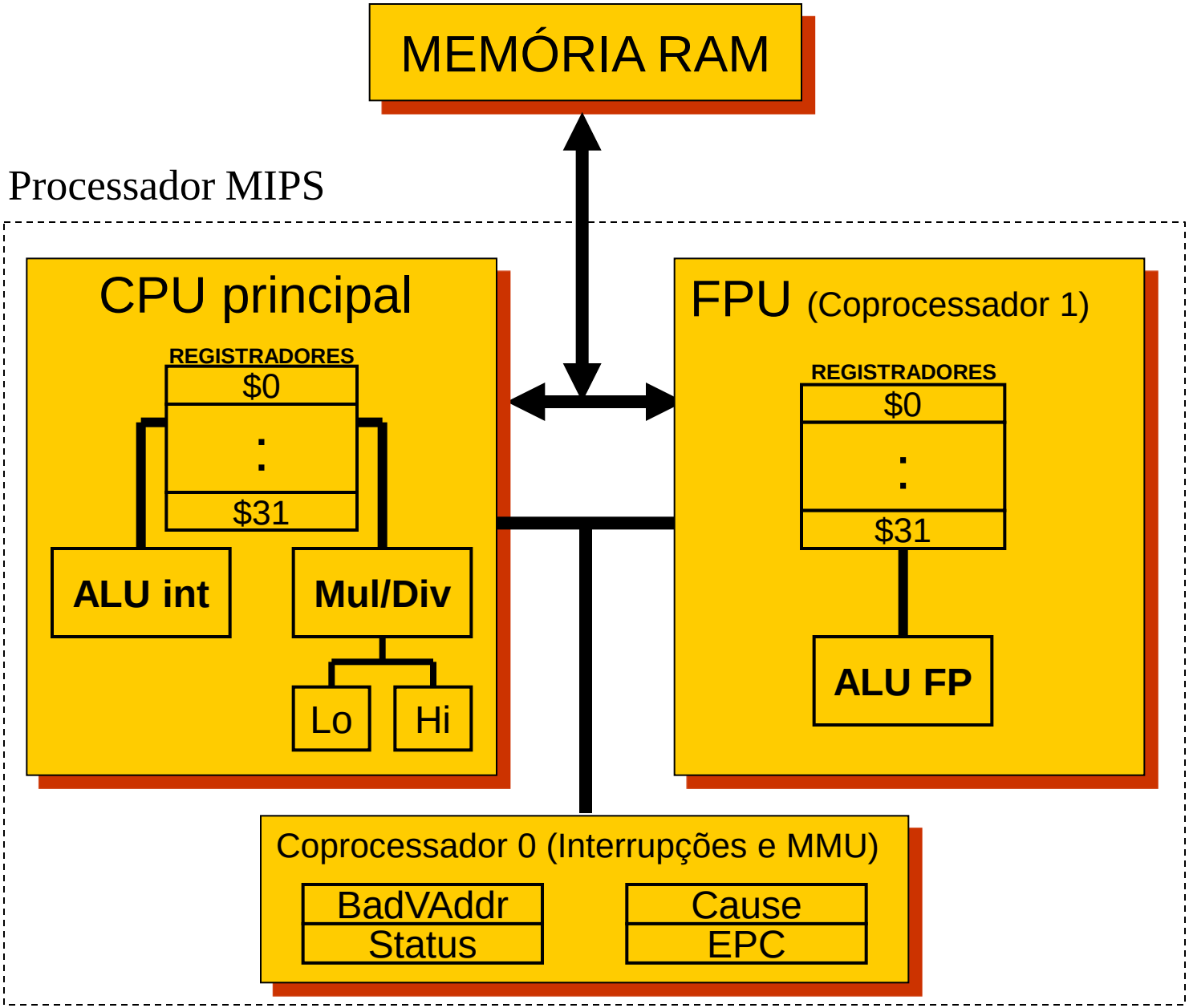
After many years of hearing only rumors about this Chinese CPU called Loongson it is finally here. And it looks like whoever is behind this CPU knows how to do a real damage when selling their product - Loongson is intended for Netbooks - high volume, low price market segment. An extremely popular market segment. Perfect for initial market penetration using low pricing as its main tool.

"The Gdium Liberty 1000 uses a 900-MHz 64-bit Loongson 2F by STMicroelectronics. Coupled with 512MB of RAM....
...Price as Reviewed: $349.99"

http://www.laptopmag.com/review/laptops/emteo-gdium-liberty-1000.aspx?page=1

Went to wiki to see what they have in that awesome beast-CPU and dug up this:

** 4-way Superscalar, Out-of-order execution, 64 bit MIPS Architecture processor core.*
** Little endian MIPS III-compatible ISA*
** 5 execution units: 2 ALUs, 2 FPUs, and 1 AGU (address generation unit)*
** SIMD unit is integrated with one of the 2 FPUs*
** Separate 64/64 KiB instruction and data L1 caches*
** on-chip 512 KiB 4-way set associative L2 cache*
** Integrated DDR2 memory controller*
** Integrated, very simple video accelerator*


MEMÓRIA RAM
Processador MIPS
CPU principal
REGISTRADORES
$0
:
$31
ALU int
Mul/Div
Lo
Hi
FPU (Coprocessador 1)
REGISTRADORES
$0
:
$31
ALU FP
Coprocessador 0 (Interrupções e MMU)
BadVAddr
Status
Cause
EPC

Organização MIPS

Um processador MIPS consiste em uma unidade central processadora de inteiros (32 ou 64 bits) (CPU) e um conjunto de coprocessadores para tarefas auxiliares.

Coprocessador 0: gerencia *traps*, interrupções, exceções, memória cache e memória virtual (MMU).

Coprocessador 1: é a unidade processadora de números em ponto flutuante (FPU).

Coprocessadores 2,3,...: Processadores de aplicação específica

Arquitetura MIPS32 - Operandos

Local do operando	Exemplo	Comentários
Banco de 32 Registradores	$zero, $at, $a0, ..., $a3, $v0, $v1 $t0,...,$t8, $s0,...,$s7, $ra, $fp, $gp, $sp	Local de acesso mais rápido a variáveis
Memória RAM	M[0], M[4], M[8],..., M[2^{32}-4] M[0], M[2], M[4],..., M[2^{32}-2] M[0], M[1], M[2],..., M[2^{32}-1]	Acesso à word, half-word ou byte
Acesso imediato	addi $t0,$t0,IMM	Local de acesso mais rápido a constantes

MIPS32: Arquitetura de 32 bits
-Registradores de 32 bits
-Endereçamento de 32 bits
-Imediato de 16 bits

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	`add $s1,$s2,$s3`	$s1 = $s2 + $s3	Three register operands
	subtract	`sub $s1,$s2,$s3`	$s1 = $s2 – $s3	Three register operands
	add immediate	`addi $s1,$s2,20`	$s1 = $s2 + 20	Used to add constants
Data transfer	load word	`lw $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Word from memory to register
	store word	`sw $s1,20($s2)`	Memory[$s2 + 20] = $s1	Word from register to memory
	load half	`lh $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load half unsigned	`lhu $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Halfword memory to register
	store half	`sh $s1,20($s2)`	Memory[$s2 + 20] = $s1	Halfword register to memory
	load byte	`lb $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Byte from memory to register
	load byte unsigned	`lbu $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Byte from memory to register
	store byte	`sb $s1,20($s2)`	Memory[$s2 + 20] = $s1	Byte from register to memory
	load linked word	`ll $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	store condition. word	`sc $s1,20($s2)`	Memory[$s2+20]=$s1;$s1=0 or 1	Store word as 2nd half of atomic swap
	load upper immed.	`lui $s1,20`	$s1 = 20 * 2^{16}	Loads constant in upper 16 bits
Logical	and	`and $s1,$s2,$s3`	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	`or $s1,$s2,$s3`	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	`nor $s1,$s2,$s3`	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	`andi $s1,$s2,20`	$s1 = $s2 & 20	Bit-by-bit AND reg with constant
	or immediate	`ori $s1,$s2,20`	$s1 = $s2 | 20	Bit-by-bit OR reg with constant
	shift left logical	`sll $s1,$s2,10`	$s1 = $s2 << 10	Shift left by constant
	shift right logical	`srl $s1,$s2,10`	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	`beq $s1,$s2,25`	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	`bne $s1,$s2,25`	if ($s1!= $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	`slt $s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for `beq`, `bne`
	set on less than unsigned	`sltu $s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than unsigned
	set less than immediate	`slti $s1,$s2,20`	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant
	set less than immediate unsigned	`sltiu $s1,$s2,20`	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant unsigned
Unconditional jump	jump	`j 2500`	go to 10000	Jump to target address
	jump register	`jr $ra`	go to $ra	For switch, procedure return
	jump and link	`jal 2500`	$ra = PC + 4; go to 10000	For procedure call

Arquitetura MIPS32 -

Convenção do uso dos Registradores...

Todos os registradores são fisicamente iguais (exceção $zero e $ra).
Logo: Convenção é o uso sugerido para padronização

O registrador **$0 ($zero) :** contém sempre o valor **0** (*hardwired*).

$1 (**$at**) : reservado para uso do montador

$2 e **$3** (**$v0**, **$v1**) : retorno de valores de funções.

$4 ... **$7** (**$a0,...,$a3**) : passagem de argumentos de funções

$8...**$15**, **$24**, **$25** (**$t0...$t9**) : dados temporários

$16...**$23** (**$s0**...**$s7**) : dados salvos

Arquitetura MIPS32 -

...Convenção do uso dos Registradores...

$26 e **$27** (**$k0, $k1**): para uso do kernel do sistema operacional

$28 ($gp): Ponteiro Global

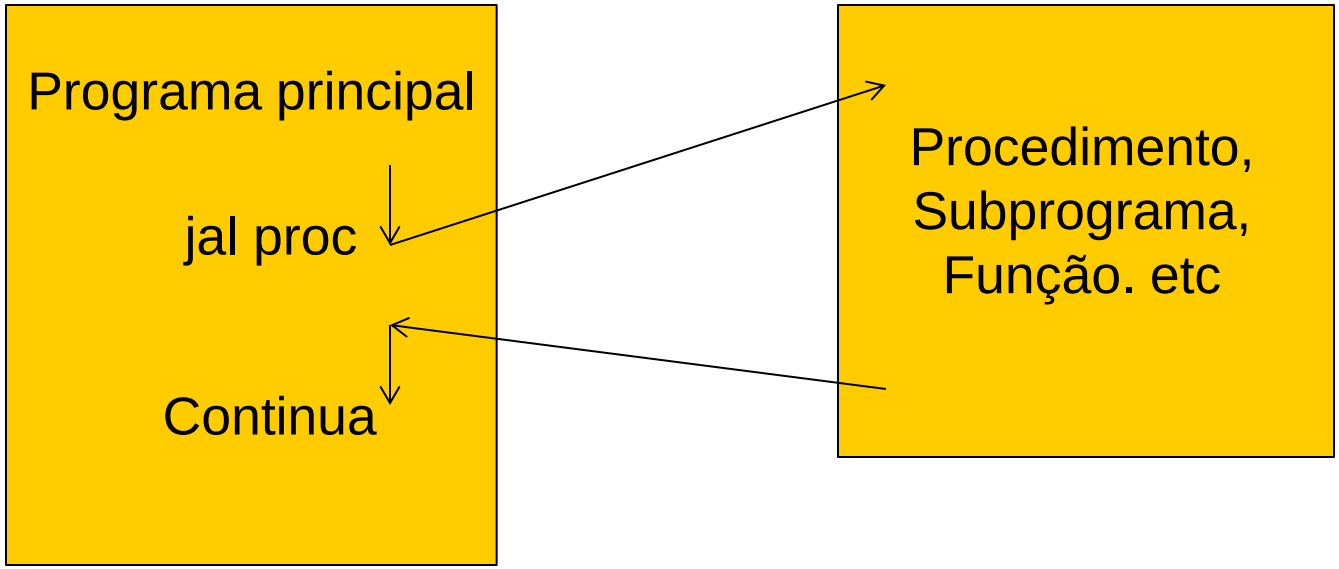
$29 ($sp): Ponteiro da Pilha

$30 ($fp): Ponteiro de Frame

$31 ($ra): Endereço de Retorno (usado pela instrução **jal**)

Arquitetura MIPS32 -

...Convenção do uso dos Registradores

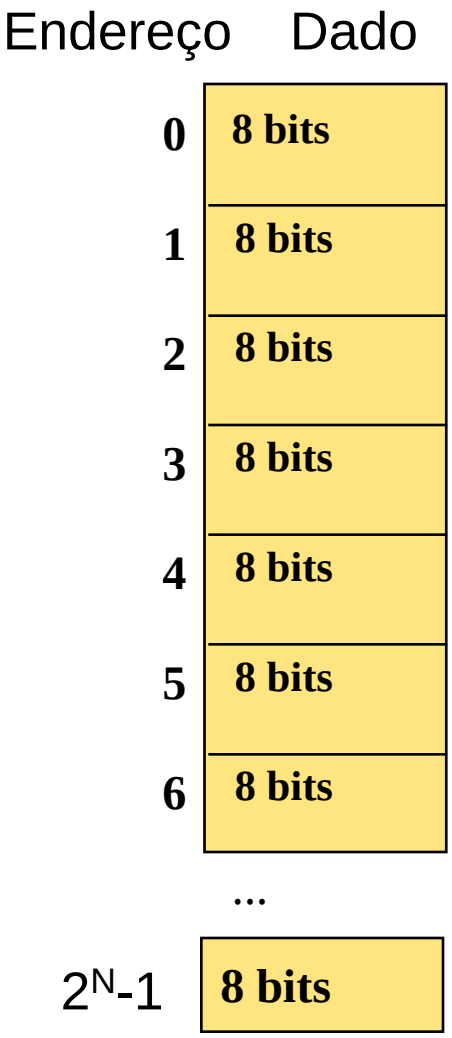


Não-preservados: $v0,$v1, $a0, $a1, $a2, $a3, $t1,.., $t9

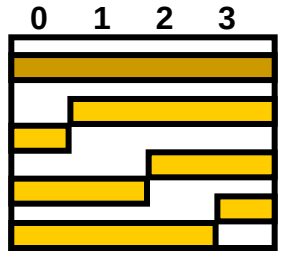
Preservados: $s0, $s9, $gp, $sp, $fp, $ra

Arquitetura MIPS32 -

Organização da memória



- Grande *array* unidimensional
- "*Byte addressing*" significa que um endereço aponta para um *byte* na memória
  - **Word: 32 bits -> 4 endereços**
  - **Half-word: 16bits-> 2 endereços**
  - **Byte: 8 bits -> 1 endereço**.



- Processador:
  - **32 bits: N=32**
  - **64 bits: N=36, 40, 48, ...,64**

Arquitetura MIPS32 -

Ordenamento de Bytes (Byte Order)

Little-endian:

Endereço: x x+1 x+2 x+3

LSB MSB

Ex.: Intel 80x86, Z80, 8051, MIPS, Vax, DEC Alpha, Atmel AVR

Big-endian:

Endereço: x x+1 x+2 x+3

MSB LSB

Ex.: IBM 360/370, Motorola 68000, HP PA, MIPS32, MIPS64

Bi-endian: MIPS(R2000), Sparc, ARM

Arquitetura MIPS32 –

Arquitetura Load/Store

- Apenas as instruções ***load*** e ***store*** têm acesso à memória.
- As outras instruções operam apenas com registradores.
- Modo de endereçamento básico do MIPS:

imm (**register**) endereço = registrador + imediato.

Formato do endereço	Endereço calculado
(register)	**Conteúdo do registrador**
imm	**Inteiro imediato**
imm (register)	**Inteiro imediato + conteúdo do registrador**
symbol	**Endereço de symbol**
symbol +/- imm	**Endereço de symbol +/- inteiro imediato**
symbol +/- imm (register)	**Endereço de symbol +/- (inteiro imediato + conteúdo do registrador)**

Arquitetura MIPS32 –

Princípios

Princípios utilizados no projeto da ISA MIPS:

- Simplicidade favorece regularidade.
- Menor significa mais rápido.
- Bons projetos exigem bons compromissos.

Objetivos de projeto de uma ISA:

maximizar o desempenho,
minimizar o custo,
reduzir o tempo de projeto.

Arquitetura MIPS32 –

Aritmética

- Instruções **tipo-R**: 3 operandos regulares
- A ordem dos operandos é fixa: Destino, Origem, Origem

Exemplos:

Código C :

```
int A,B,C;
...
A = B + C;
```

Código MIPS :

```
add $s0, $s1, $s2
```

- Registradores são associados às variáveis pelo compilador

Código C:

```
A = B + C + D;
E = F - A;
```

Código MIPS:

```
add $t0, $s1, $s2
add $s0, $t0, $s3
sub $s4, $s5, $s0
```

Arquitetura MIPS32 –

Acesso à memória

- Instruções:

	LOAD	STORE
word :	lw $s0,10($t1)	sw $s0,10($t1)
half-word:	lh $s0,10($t1)	sh $s0,10($t1)
byte:	lb $s0,10($t1)	sb $s0,10($t1)

- Registradores são associados às variáveis pelo compilador
- Exemplo:

```
                        int h, A[100];
                        ....
Código C:               A[12] = h + A[8];

Código MIPS:            lw $t0, 32($s3)
                        add $t0, $s1, $t0
                        sw $t0, 48($s3)
```

Arquitetura MIPS32 –

Operador Imediato

- Instruções **tipo-I**: Imediato
- É comum a operação com constantes, logo agilize!
- Exemplo:

```
                        int A,B;
                        ....
Código C:               B++;
                        A=B-37;

Código MIPS:            addi $s1, $s1, 1
                        addi $s2,$s1,-37
```

Obs.: Não existe subi !

Exemplo de procedimento

Compilar o código:

```
swap(int v[], int k)
{ int temp;
      temp = v[k];
      v[k] = v[k+1];
      v[k+1] = temp;
}
```

```
swap:
   muli $t0, $a1, 4     # calcula o offset em bytes
   add $t0, $a0, $t0    # calcula o endereço em bytes
   lw $t1, 0($t0)       # lê o valor v[k]
   lw $t2, 4($t0)       # lê o valor v[k+1]
   sw $t2, 0($t0)       # escreve em v[k]
   sw $t1, 4($t0)       # escreve em v[k+1]
   jr $ra               # retorna da função
```