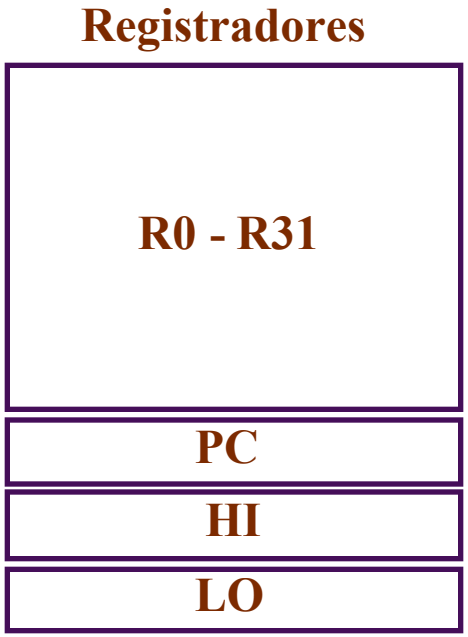
Universidade de Brasília
Departamento de Ciência da Computação

Trabalho 2

ISA do MIPS (simplificada)

- Categorias de Instruções:
  - Aritmética
  - Transferência de Dados
  - Lógica
  - Desvio condicional
  - Desvio incondicional



- 3 Formatos de Instrução: 32 bits

Nome	Número	Uso (sugerido)
zero	0	Constante 0
at	1	Reservado para o montador
v0	2	Avaliação de expressão e resultado de função
v1	3	Avaliação de expressão e resultado de função
a0	4	Argumento 1
a1	5	Argumento 2
a2	6	Argumento 3
a3	7	Argumento 4
t0	8	Temporário (não preservado após *call*)
t1	9	Temporário (não preservado após *call*)
t2	10	Temporário (não preservado após *call*)
t3	11	Temporário (não preservado após *call*)
t4	12	Temporário (não preservado após *call*)
t5	13	Temporário (não preservado após *call*)
t6	14	Temporário (não preservado após *call*)
t7	15	Temporário (não preservado após *call*)

Nome	Número	Uso (sugerido)
s0	16	Temporário salvo (preservado após *call*)
s1	17	Temporário salvo (preservado após *call*)
s2	18	Temporário salvo (preservado após *call*)
s3	19	Temporário salvo (preservado após *call*)
s4	20	Temporário salvo (preservado após *call*)
s5	21	Temporário salvo (preservado após *call*)
s6	22	Temporário salvo (preservado após *call*)
s7	23	Temporário salvo (preservado após *call*)
t8	24	Temporário (não preservado após *call*
t9	25	Temporário (não preservado após *call*
k0	26	Reservado para o núcleo do sistema operacional
k1	27	Reservado para o núcleo do sistema operacional
gp	28	Ponteiro para variáveis global
sp	29	Ponteiro de pilha
fp	30	Ponteiro de *frame*
ra	31	Endereço de retorno (utilizado por chamada de sub-programa)

Registradores

```
enum REGISTERS {
   ZERO=0, AT=1,  V0=2,
   V1=3,   A0=4,  A1=5,
   A2=6,   A3=7,  T0=8,
   T1=9,   T2=10, T3=11,
   T4=12,  T5=13, T6=14,
   T7=15,  T8=24, T9=25,
   S0=16,  S1=17, S2=18,
   S3=19,  S4=20, S5=21,
   S6=22,  S7=23, K0=26,
   K1=27,  GP=28, SP=29,
   FP=30,  RA=31 };
```

Categoria	Instrução	Exemplo	Significado	Comentários
Aritmética	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Três operandos; dados nos registradores
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	Três operandos; dados nos registradores
	add immediate	addi $s1,$s2,20	$s1 = $s2 + 20	Usada para somar constantes
Transferência de dados	load word	lw $s1,20($s2)	$s1 = Memória[$s2 + 20]	Dados da memória para o registrador
	store word	sw $s1,20($s2)	Memória[$s2 + 20] = $s1	Dados do registrador para a memória
	load half	lh $s1,20($s2)	$s1 = Memória[$s2 + 20]	Halfword da memória para registrador
	load half unsigned	lhu $s1,20($s2)	$s1 = Memória[$s2 + 20]	Halfword da memória para registrador
	store half	sh $s1,20($s2)	Memória[$s2 + 20] = $s1	Halfword de um registrador para memória
	load byte	lb $s1,20($s2)	$s1 = Memória[$s2 + 20]	Byte da memória para registrador
	load byte unsigned	lbu $s1,20($s2)	$s1 = Memória[$s2 + 20]	Byte da memória para registrador
	store byte	sb $s1,20($s2)	Memória[$s2 + 20] = $s1	Byte de um registrador para memória
	load linked word	ll $s1,20($s2)	$s1 = Memória[$s2 + 20]	Carrega word como 1ª metade do swap atômico
	store condition, word	sc $s1,20($s2)	Memória[$s2+20]=$s1;$s1=0 or 1	Armazena word como 2ª metade do swap atômico
	load upper immed.	lui $s1,20	$s1 = 20 * 2^{16}	Carrega constante nos 16 bits mais altos
Lógica	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Três operadores em registrador; AND bit a bit
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Três operadores em registrador; OR bit a bit
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 | $s3)	Três operadores em registrador; NOR bit a bit
	and immediate	andi $s1,$s2,20	$s1 = $s2 & 20	AND bit a bit registrador com constante
	or immediate	ori $s1,$s2,20	$s1 = $s2 | 20	OR bit a bit registrador com constante
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Deslocamento à esquerda por constante
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Deslocamento à direita por constante
Desvio condicional	branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Testa igualdade; desvio relativo ao PC
	branch on not equal	bne $s1,$s2,25	if ($s1!= $s2) go to PC + 4 + 100	Testa desigualdade; relativo ao PC
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compara menor que; usado com beq, bne
	set on less than unsigned	sltu $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compara menor que sem sinal
	set less than immediate	slti $s1,$s2,20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compara menor que constante
	set less than immediate unsigned	sltiu $s1,$s2,20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compara menor que constante sem sinal
Desvio incondicional	jump	j 2500	go to 10000	Desvia para endereço de destino
	jump register	jr $ra	go to $ra	Para switch e retorno de procedimento
	jump and link	jal 2500	$ra = PC + 4; go to 10000	Para chamada de procedimento

Tipos variáveis C / C++

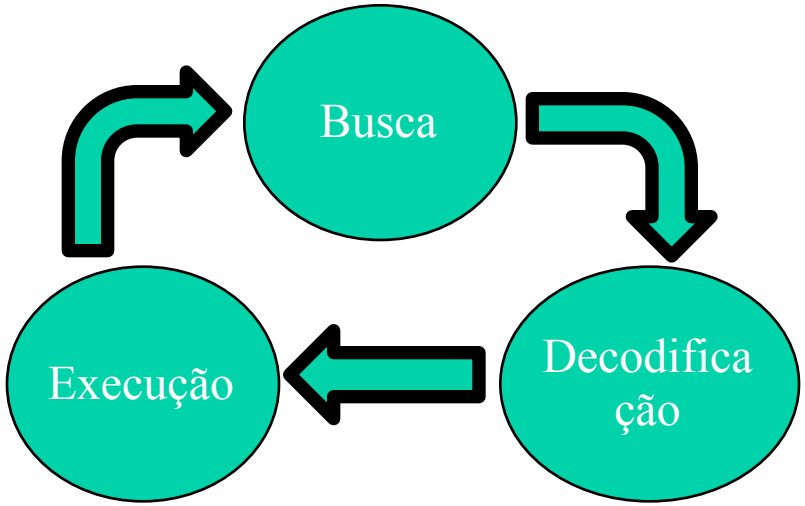
→ stdint.h

int8_t	uint8_t
int16_t	uint16_t
int32_t	uint32_t
int64_t	uint64_t

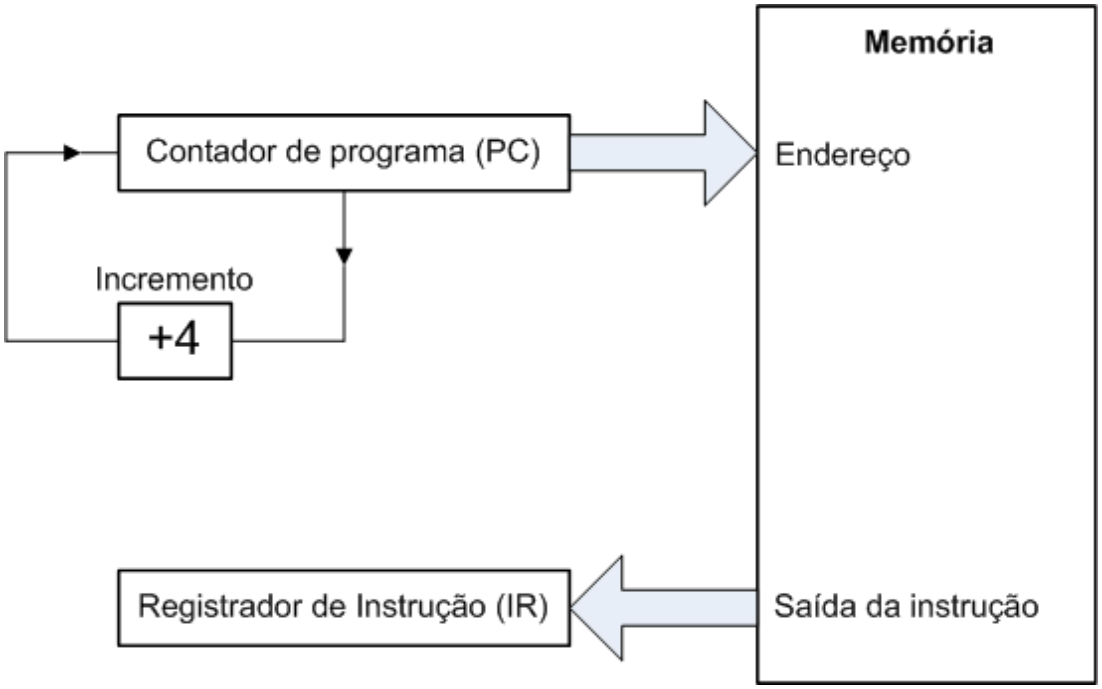
Programa armazenado (conceito)

Ciclos de busca e execução:

- Instruções são buscadas e colocadas num registrador especial (IR – *Instruction Register*).
- Bits deste registrador "controlam" as ações subseqüentes necessárias à execução da instrução.
- Busca a próxima instrução e continua...

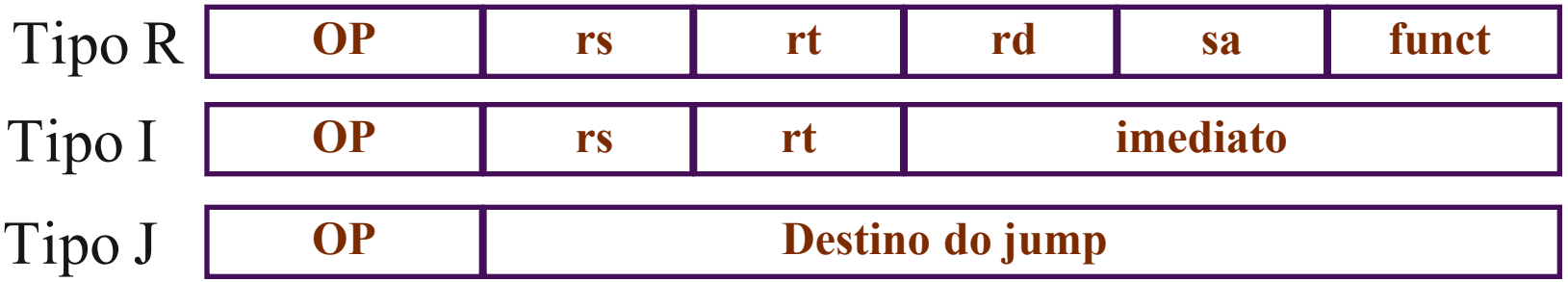


Busca



- **PC**: registrador que contém o endereço da próxima intrução que será lida e executada
- **IR**: armazena os bits relativos à instrução atual

Decodificação




```
void decode ()
{
   opcode = (ri >> 26) & 0x3F;
   ...
}
```

Decodificação

```
enum OPCODES {    // lembrem que só são considerados os 6 primeiros bits!!

    EXT=0x00,     LW=0x23,      LB=0x20,      LBU=0x24,
    LH=0x21,      LHU=0x25,     LUI=0x0F,     SW=0x2B,
    SB=0x28,      SH=0x29,      BEQ=0x04,     BNE=0x05,
    BLEZ=0x06,    BGTZ=0x07,    ADDI=0x08,    ADDIU=0x09,
    SLTI=0x0A,    SLTIU=0x0B,   ANDI=0x0C,    ORI=0x0D,
    XORI=0x0E,    J=0x02,       JAL=0x03
};

enum FUNCT {
    ADD=0x20,     SUB=0x22,     MULT=0x18,    DIV=0x1A,     AND=0x24,
    OR=0x25,      XOR=0x26,     NOR=0x27,     SLT=0x2A,     JR=0x08,
    SLL=0x00,     SRL=0x02,     SRA=0x03,     SYSCALL=0x0c,   MFHI=0x10, MFLO=0x12
};
```

Decodificação

http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html

https://en.wikipedia.org/wiki/MIPS_instruction_set

Cuidado com extensão do sinal

```
opcode  = (ri >> 26) & 0x3F;
```

Se ri = 0xFFFFFFFF e for do tipo int32_t

```
opcode   = (ri >> 26) = 0xFFFFFFFF = -1

opcode   = ((uint32_t)ri >> 26) = 0x0000003F
```

Execução

Verifica-se o opcode/funct da instrução executada

Define-se registradores/imediato/endereço jump

Executa a instrução

```
void execute()
{
    switch (opcode) {
        case EXT:
            switch (funct) {
                case ADD:  breg[rd] = breg[rs] + breg[rt]; break;
}
```

Instruções tipo LB e LH

Ponteiros!!

```
int32_t a[2];
int32_t b;

a[0] = 0x03020100;
a[1] = 0xfffffffe;

b = *((int16_t *)&a + 0); → b = 0x00000100

b = *((int16_t *)&a + 1); → b = 0x00000302

b = *((int16_t *)&a + 2); → b = 0xfffffffe

b = *((uint16_t *)&a + 3); → b = 0x0000ffff
```