DrMIPS – Manual de Configuração

Bruno Nova

17 de Setembro de 2015

Conteúdo

Capítulo 1

Introdução

O DrMIPS fornece vários caminhos de dados uniciclo e *pipeline* do MIPS. Estes caminhos de dados são definidos em ficheiros JSON (`http://json.org/`), e têm a extensão `.cpu`. Estes ficheiros de CPU podem ser modificados, e novos podem ser criados.

Os conjuntos de instruções usados pelos caminhos de dados são também definidos em ficheiros JSON, usando a extensão `.set`. Estes podem também ser criados e modificados.

Este manual explica, com algum detalhe, a sintaxe de ambos estes ficheiros. O Capítulo 2 explica a sintaxe dos ficheiros de CPU enquanto que o Capítulo 3 explica a sintaxe dos ficheiros de instruções.

Capítulo 2

Ficheiro de CPU

As diferentes versões do processador MIPS são definidas em ficheiros de CPU. Estes podem ser editados/configurados e novos podem ser criados. Este capítulo explica a sintaxe destes ficheiros.

Os ficheiros de CPU são formatados em JSON. Um exemplo parcial de um ficheiros de CPU é mostrado abaixo:

```
1  {
2    "components": {
3      "MUX_DST": {"type": "mux", "x": 205, "y": 260, "size": 5, "
           sel": "RegDst", "out": "OUT", "in": ["0", "1"], "desc":
            {"default": "Selects rt or rd as the destination
           register.", "pt": "Selecciona o rt ou rd como registo
           de destino."}},
4      ...
5    },
6    "wires": {
7      {"from": "DIST_INST", "out": "15-11", "to": "MUX_DST", "in
           ": "1", "start": {"x": 185, "y": 270}, "points": [{"x":
            195, "y": 270}, {"x": 195, "y": 282}]},
8      {"from": "MUX_DST", "out": "OUT", "to": "REG", "in": "
           WriteReg", "end": {"x": 250, "y": 277}},
9      ...
10   },
11   "reg_names": ["zero", "at", "v0", "v1", "a0", "a1", "a2", "a3
         ", "t0", "t1", "t2", "t3", "t4", "t5", "t6", "t7", "s0",
         "s1", "s2", "s3", "s4", "s5", "s6", "s7", "t8", "t9", "k0
         ", "k1", "gp", "sp", "fp", "ra"],
12   "instructions": "default.set"
13 }
```

As várias secções que compõe os ficheiros de CPU são detalhadas nas seguintes secções deste capítulo.

2.1 components

Esta secção define todos os componentes do processador e suas propriedades. Um exemplo da definição de um componente é mostrado abaixo:

```
1 "MUX_DST": {"type": "mux", "x": 205, "y": 260, "size": 5, "sel
      ": "RegDst", "out": "OUT", "in": ["0", "1"], "desc": {"
      default": "Selects rt or rd as the destination register.",
      "pt": "Selecciona o rt ou rd como registo de destino."}}
```

Cada componente é identificado por um ID único (`MUX_DST` neste exemplo). As propriedades do componente são definidas entre chavetas como um objecto JSON. Muitas propriedades são específicas a cada tipo de componente, mas algumas existem para todos os componentes. Estas são:

- **type**: O tipo de componente. O componente no exemplo é um multiplexador. Os diferentes tipos de componentes são explicados a seguir.
- **latency**: Opcional. Um inteiro com a latência do componente em ps. A latência por omissão é 0 ps.
- **x**: A coordenada x do canto superior esquerdo do componente no caminho de dados gráfico. O valor mínimo é 0 e corresponde à borda da esquerda do caminho de dados. Não há valor máximo.
- **y**: A coordenada y do canto superior esquerdo do componente no caminho de dados gráfico. O valor mínimo é 0 e corresponde à borda de cima do caminho de dados. Não há valor máximo.
- **desc**: Opcional. Descrição específica do componente, em cada idioma, mostrada na dica de ajuda do componente. O valor é um objecto JSON onde a descrição para cada idioma é definida na forma `"código_idioma": "Descrição."`. O identificador `código_idioma` é o código do idioma, como `pt` ou `pt_BR`. O idioma especial `default` deverá definir a descrição por omissão em Inglês que é usada quando uma descrição específica para o idioma não está disponível.

As subsecções seguintes explicam os diferentes tipos de componentes disponíveis e as suas propriedades específicas. Os títulos das subsecções são os valores que devem ser escritos na propriedade `type` dos componentes. Esta propriedade não é sensível à capitalização.

2.1.1 Add

Um somador que soma os valores das entradas. As propriedades específicas são:

- **in1**: Identificador da primeira entrada.
- **in2**: Identificador da segunda entrada.
- **out**: Identificador da saída.

2.1.2 ALU

A ALU básica. Só uma ALU ou ALU Estendida pode estar presente. As propriedades específicas são:

- **in1**: Identificador da primeira entrada.
- **in2**: Identificador da segunda entrada.
- **control**: Identificador da entrada de controlo (que selecciona a operação a realizar).
- **out**: Identificador da saída de resultado.
- **zero**: Identificador da saída de 1 bit do *zero*.

2.1.3 ALU_Control

O componente que controla a ALU. Só um pode estar presente. As propriedades específicas são:

- **aluop**: Identificador da entrada $ALUOp$.
- **func**: Identificador da entrada *func* (do campo de função da instrução).

2.1.4 And

Uma porta lógica *AND*. As propriedades específicas são:

- **in1**: Identificador da primeira entrada.
- **in2**: Identificador da segunda entrada.
- **out**: Identificador da saída.

2.1.5 Concat

Um "concatenador" que concatena os valores de duas entradas numa única saída. O valor da saída é a concatenação do valor da primeira entrada (como bits mais significativos) com o valor da segunda entrada (como bits menos significativos). O tamanho da saída é igual à soma dos tamanhos das entradas. As propriedades específicas são:

- **in1**: Identificador da primeira entrada.

- **in2**: Identificador da segunda entrada.
- **out**: Identificador da saída.

As propriedades de ambas as entradas são definidas como objectos JSON. As propriedades destes objectos são:

- **id**: Identificador da entrada.
- **size**: Tamanho da entrada (em bits).

2.1.6 Const

Um componente que produz um valor constante. As propriedades específicas são:

- **out**: Identificador da saída.
- **val**: O valor constante.
- **size**: Tamanho da saída (em bits).

2.1.7 Control

A unidade de controlo. O caminho de dados tem de ter uma unidade de controlo. Este componente tem apenas uma propriedade específica: **in**, que é o identificador da entrada.

2.1.8 Dist

Este componente distribui os bits da entrada por várias saídas. As propriedades específicas são:

- **in**: Propriedades da entrada como um objecto JSON. As propriedades do objecto são:
  - **id**: Identificador da entrada.
  - **size**: Tamanho da entrada (em bits).
- **out**: Propriedades das saídas como um *array* JSON. Cada elemento do *array* define as propriedades de uma saída como um objecto JSON. As propriedades dos objectos do *array* são:
  - **msb**: Índice do bit mais significativo da entrada.
  - **lsb**: Índice do bit menos significativo da entrada.
  - **id**: Opcional. Identificador da saída. Se omitido, o identificador corresponde a "`<msb>-<lsb>`".

2.1.9 DMem

A memória de dados. Só uma pode estar presente. As propriedades específicas são:

- **size**: Tamanho da memória (número de posições de memória de 32 bits).
- **address**: Identificador da entrada *Address*.
- **write_data**: Identificador da entrada *WriteData*.
- **out**: Identificador da saída.
- **mem_read**: Identificador da entrada de controlo *MemRead*.
- **mem_write**: Identificador da entrada de controlo *MemWrite*.

2.1.10 Ext_ALU

Uma ALU estendida. Esta ALU armazena os registos `hi` e `lo` e é capaz de realizar multiplicações e divisões. Apenas uma ALU ou ALU Estendida pode estar presente. As propriedades específicas são as mesmas da ALU básica:

- **in1**: Identificador da primeira entrada.
- **in2**: Identificador da segunda entrada.
- **control**: Identificador da entrada de controlo (que selecciona a operação a realizar).
- **out**: Identificador da saída de resultado.
- **zero**: Identificador da saída de 1 bit do *zero*.

2.1.11 Fork

Este componente divide uma ligação em várias outras ligações com o mesmo tamanho. As propriedades específicas são:

- **in**: Identificador da entrada.
- **size**: Tamanho da entrada e das saídas (em bits).
- **out**: *Array* com os identificadores das saídas.

2.1.12 Fwd_Unit

A unidade de atalhos da *pipeline*. Apenas uma pode estar presente. As propriedades específicas são:

- **ex_mem_reg_write**: Identificador da entrada *EX/MEM.RegWrite*.
- **mem_wb_reg_write**: Identificador da entrada *MEM/WB.RegWrite*.
- **ex_mem_rd**: Identificador da entrada *EX/MEM.Rd*.
- **mem_wb_rd**: Identificador da entrada *MEM/WB.Rd*.
- **id_ex_rs**: Identificador da entrada *ID/EX.Rs*.
- **id_ex_rt**: Identificador da entrada *ID/EX.Rt*.
- **fwd_a**: Identificador da saída *ForwardA*.
- **fwd_b**: Identificador da saída *ForwardB*.

2.1.13 Hzd_Unit

A unidade de detecção de conflitos da *pipeline*. Apenas uma pode estar presente. As propriedades específicas são:

- **id_ex_mem_read**: Identificador da entrada *ID/EX.MemRead*.
- **id_ex_rt**: Identificador da entrada *ID/EX.Rt*.
- **if_id_rs**: Identificador da entrada *IF/ID.Rs*.
- **if_id_rt**: Identificador da entrada *IF/ID.Rt*.
- **stall**: Identificador da saída.

2.1.14 IMem

A memória de instruções. O caminho de dados tem de ter uma memória de instruções. As propriedades específicas são:

- **in**: Identificador da entrada.
- **out**: Identificador da saída.

2.1.15 Mux

Um multiplexador. As propriedades específicas são:

- **size**: O tamanho das entradas e saída (em bits).
- **sel**: Identificador da entrada de selecção.
- **out**: Identificador da saída.
- **in**: *Array* com os identificadores das saídas.

2.1.16 Not

Uma porta lógica *NOT*. As propriedades específicas são:

- **in**: Identificador da entrada.
- **out**: Identificador da saída.

2.1.17 Or

Uma porta lógica *OR*. As propriedades específicas são:

- **in1**: Identificador da primeira entrada.
- **in2**: Identificador da segunda entrada.
- **out**: Identificador da saída.

2.1.18 PipeReg

Um registo de *pipeline* que separa duas etapas da *pipeline*. Um caminho de dados *pipeline* tem de ter exactamente 4 destes registos (correspondentes a um *pipeline* de 5 etapas). Adicionalmente, os identificadores destes componentes têm de ser: `IF/ID`, `ID/EX`, `EX/MEM` and `MEM/WB`. As propriedades específicas são:

- **regs**: Definição dos registos guardados. O valor é um objecto JSON onde cada propriedade define um registo: o identificador é o identificador do registo e correspondente entrada e saída, e o valor é o tamanho do registo (em bits).
- **flush**: Opcional. Identificador da entrada de controlo *Flush*.
- **write**: Opcional. Identificador da entrada de controlo *Write*.

2.1.19 PC

O contador do programa. O caminho de dados tem de ter um contador do programa. As propriedades específicas são:

- **in**: Identificador da entrada.
- **out**: Identificador da saída.
- **write**: Opcional. Identificador da saída de controlo *Write*.

2.1.20 RegBank

O banco de registos. O caminho de dados tem de ter um banco de registos. As propriedades específicas são:

- **num_regs**: O número de registos. Tem de ser maior do que 1 e uma potência de 2.
- **read_reg1**: Identificador da entrada *ReadReg1*.
- **read_reg2**: Identificador da entrada *ReadReg2*.
- **read_data1**: Identificador da saída *ReadData1*.
- **read_data2**: Identificador da saída *ReadData2*.
- **write_reg**: Identificador da entrada *WriteReg*.
- **write_data**: Identificador da entrada *WriteData*.
- **reg_write**: Identificador da entrada de controlo *RegWrite*.
- **forwarding**: Opcional. Se `true`, o banco de registos irá usar atalhos internos (para caminhos de dados *pipeline*).
- **const_regs**: Opcional. *Array* JSON que define os registos constantes. Cada elemento é ou o índice do registo ou um objecto JSON com as seguintes propriedades:
  - **reg**: Índice do registo.
  - **val**: O valor constante do registo.

2.1.21 SExt

Uma extensão de sinal. As propriedades específicas são:

- **in**: Propriedades da entrada.
- **out**: Propriedades da saída.

As propriedades da entrada e da saída são definidas como objectos JSON. As propriedades destes objectos são:

- **id**: Identificador da entrada/saída.
- **size**: Tamanho da entrada/saída (em bits).

2.1.22 SLL

Um deslocador para a esquerda. As propriedades específicas são:

- **in**: Propriedades da entrada.
- **out**: Propriedades da saída.
- **amount**: Número de bits a deslocar

As propriedades da entrada e da saída são definidas como objectos JSON. As propriedades destes objectos são:

- **id**: Identificador da entrada/saída.
- **size**: Tamanho da entrada/saída (em bits).

2.1.23 Xor

Uma porta lógica *XOR*. As propriedades específicas são:

- **in1**: Identificador da primeira entrada.
- **in2**: Identificador da segunda entrada.
- **out**: Identificador da saída.

2.1.24 ZExt

Uma extensão de valor. As propriedades específicas são:

- **in**: Propriedades da entrada.
- **out**: Propriedades da saída.

As propriedades da entrada e da saída são definidas como objectos JSON. As propriedades destes objectos são:

- **id**: Identificador da entrada/saída.
- **size**: Tamanho da entrada/saída (em bits).

2.2 wires

Esta secção define todas as ligações que conectam os componentes do CPU. Um exemplo da definição de uma ligação é mostrado abaixo:

```
1 {"from": "DIST_INST", "out": "15-11", "to": "MUX_DST", "in":
    "1", "start": {"x": 185, "y": 270}, "points": [{"x": 195, "
    y": 270}, {"x": 195, "y": 282}], "end": {"x": 205, "y":
    282}}
```

Cada ligação conecta uma saída de um componente à entrada de outro componente. Uma ligação é definida como um objecto JSON com várias propriedades. Estas são:

- **from**: O ID do componente de onde a conexão é feita (origem).
- **out**: O ID da saída do componente de origem de onde a conexão é feita.
- **to**: O ID do componente para onde a conexão é feita (destino).
- **in**: O ID da entrada do componente de destino para onde a conexão é feita.
- **start**: Opcional. Um objecto JSON que define a posição gráfica de início no caminho de dados, se a posição por omissão não é a desejável.
- **points**: Opcional. Um *Array* de objectos JSON que definem as posições dos pontos intermédios da ligação, se desejado.
- **end**: Opcional. Um objecto JSON que define a posição gráfica final no caminho de dados, se a posição por omissão não é a desejável.

As posições usadas nas propriedades `start`, `points` and `end` acima são objectos JSON com duas propriedades inteiras: `x` e `y`. Cada entrada e saída de cada componente é, por omissão, "anexada" a um dos quatro lados do componente. As posições das entradas e saídas no caminho de dados e, portanto, as posições de início e fim das ligações, são calculadas automaticamente[1] mas podem ser sobrescritas pelas propriedades `start` e `end`.

2.3 reg_names

Esta secção **opcional** define os nomes "amigáveis" dos registos (i.e. `$zero`, `$t0`, etc.). O valor é um *array* de *strings* que definem os nomes dos registos,

[1] As entradas e saídas em cada lado do componente são, por omissão, ordenadas alfabeticamente pelos seus identificadores.

desde o registo `$0` até ao último, **sem o símbolo do dólar inicial**. Os registos podem sempre ser referenciados pelos seus índices (`$0`, `$1`, etc.) no simulador.

2.4 instructions

Esta secção declara o conjunto de instruções que o processador usa. O valor é o caminho **relativo** para ficheiro de instruções desejado. Estes ficheiros são explicados no próximo capítulo.

Capítulo 3

Ficheiro de instruções

Os conjuntos de instruções usados pelas diferentes versões do processador MIPS são definidos em ficheiros de instruções. Estes podem ser editados/-configurados e novos podem ser criados. Este capítulo explica a sintaxe destes ficheiros.

Tal como os ficheiros de CPU, os ficheiros de instruções são formatados em JSON. Um exemplo parcial de um ficheiro de instruções é mostrado abaixo:

```
{
  "types": {
    "R": [{"id": "op", "size": 6}, {"id": "rs", "size": 5}, {"
        id": "rt", "size": 5}, {"id": "rd", "size": 5}, {"id":
        "shamt", "size": 5}, {"id": "func", "size": 6}],
    "I": [{"id": "op", "size": 6}, {"id": "rs", "size": 5}, {"
        id": "rt", "size": 5}, {"id": "imm", "size": 16}],
    "J": [{"id": "op", "size": 6}, {"id": "target", "size":
        26}]
  },
  "instructions": {
    "add":  {"type": "R", "args": ["reg", "reg", "reg"], "
        fields": {"op": 0, "rs": "#2", "rt": "#3", "rd": "#1",
        "shamt": 0, "func": 32}, "desc": "$t1 = $t2 + $t3"},
    ...
  },
  "pseudo": {
    "move": {"args": ["reg", "reg"], "to": ["add #1, #2, $0"],
        "desc": "$t1 = $t2"},
    ...
  },
  "control": {
    "0": {"RegDst": 1, "RegWrite": 1, "ALUOp": 2, "ALUSrc": 0,
        "MemToReg": 0},
    ...
  },
  "alu": {
```

```
20      "aluop_size": 2,
21      "func_size": 6,
22      "control_size": 3,
23      "control": [
24        {"aluop": 0, "out": {"control": 2}},
25        {"aluop": 2, "func": 32, "out": {"control": 2}},
26        ...
27      ],
28      "operations": {
29        "2": "add",
30        ...
31      }
32    }
33  }
```

As várias secções que compõe os ficheiros de instruções são detalhadas nas seguintes secções deste capítulo.

3.1 types

Esta secção define os tipos de instruções existentes e os seus campos. Todas as instruções têm um tamanho de 32 bits. Um exemplo da definição de um tipo de instrução é mostrado abaixo:

```
1 "R": [{"id": "op", "size": 6}, {"id": "rs", "size": 5}, {"id":
      "rt", "size": 5}, {"id": "rd", "size": 5}, {"id": "shamt",
      "size": 5}, {"id": "func", "size": 6}]
```

Cada tipo de instrução é definido pelo seu identificador (`R` neste exemplo). Os campos de uma instrução são definidos num *array* JSON. Cada campo individual é um objecto JSON e define o identificador do campo e seu tamanho em bits. O primeiro campo neste exemplo é o campo `op` com um tamanho de 6 bits. O primeiro campo das instruções é considerado o *opcode* e tem de ter o mesmo tamanho em todos os tipos.

3.2 instructions

Esta secção define as instruções disponíveis e como elas são codificadas. Um exemplo da definição de uma instrução é mostrado abaixo:

```
1 "add":  {"type": "R", "args": ["reg", "reg", "reg"], "fields":
      {"op": 0, "rs": "#2", "rt": "#3", "rd": "#1", "shamt": 0, "
      func": 32}, "desc": "$t1 = $t2 + $t3"}
```

Cada instrução é identificada pela sua mnemónica única (`add` neste exemplo), Uma instrução é definida como um objecto JSON com várias propriedades. Estas são:

- **type**: O tipo da instrução.
- **args**: Um *array* com os tipos de cada argumento (pode ser omitido se a instrução não tiver argumentos). Os diferentes tipos de argumentos são:
  - **reg**: Um registo.
  - **int**: Um valor inteiro. Pode ser também uma etiqueta (permitido devido à pseudo-instrução `la`).
  - **target**: Uma etiqueta no código ou directamente o índice de uma instrução para um salto.
  - **offset**: Uma etiqueta no código ou directamente um *offset* de instruções para um *branch*.
  - **label**: Uma etiqueta no código ou segmento de dados ou directamente um endereço/índice.
  - **data**: Uma etiqueta no segmento de dados ou directamente um endereço mais um registo de *offset* para uma instrução de *load* ou *store* (um argumento como `label($t0)`). O utilizador pode omitir o registo de *offset*.
- **fields**: Um objecto JSON que define os valores dos campos (os campos definidos no tipo da instrução). O valor de cada campo pode ser um inteiro constante ou vir de um argumento (especificado como `"#1"`, `"#2"`, etc.). Para valores que venham de um argumento `data` é necessário especificar se eles vêm de um endereço base ou de um registo de *offset*. Isto é feito afixando `.base` ou `.offset` à referência para o argumento (`#1`, `#2`, etc.).

  Neste exemplo: `op` tem o valor constante `0`, `rs` tem o valor do 2º argumento, `rt` tem o valor do 3º argumento, etc.
- **desc**: Opcional. Uma *string* que deverá conter uma curta descrição simbólica do que a instrução faz para o utilizador ver.

3.3 pseudo

Esta secção define as pseudo-instruções disponíveis. Um exemplo da definição de uma pseudo-instrução é mostrado abaixo:

```
1 "subi": {"args": ["reg", "reg", "int"], "to": ["li $1, #3", "
      sub #1, #2, $1"], "desc": "$t1 = $t2 - 23"}
```

Cada pseudo-instrução é identificada pela sua mnemónica única (`subi` neste exemplo). Cada mnemónica apenas pode identificar uma instrução ou uma pseudo-instrução, não as duas. Uma pseudo-instrução é definida como um objecto JSON com várias propriedades. Estas são:

- **args**: Um *array* com os tipos de cada argumento (pode ser omitido se a pseudo-instrução não tiver argumentos). Estes tipos são os mesmos que estão disponíveis para as instruções, explicados na secção anterior.
- **to**: Um *array* que lista as instruções reais para as quais a pseudo-instrução é convertida quando assemblada. Os valores dos argumentos especificados pelo utilizador podem ser referenciados usando `#1` para o 1º argumento, `#2` para o 2º, e por assim adiante.
- **desc**: Opcional. Uma *string* que deverá conter uma curta descrição simbólica do que a pseudo-instrução faz para o utilizador ver.

3.4 control

Esta secção define como a unidade de controlo funciona. Mais especificamente, define os valores das saídas (sinais de controlo) para cada valor possível na entrada (o *opcode*). Um exemplo da definição dos valores dos sinais de control para um *opcode* é mostrado abaixo:

```
1 "0": {"RegDst": 1, "RegWrite": 1, "ALUOp": 2, "ALUSrc": 0, "
      MemToReg": 0}
```

Os valores dos sinais de controlo para um certo *opcode* são definidos como um objecto JSON. Os valores estão em formato decimal e os tamanhos (em bits) dos sinais de controlo são determinados automaticamente. Sinais de controlo que têm o valor `0` podem ser omitidos.

3.5 alu

Esta secção define como a ALU e o controlador da ALU funcionam. Um exemplo parcial desta secção é mostrado abaixo:

```
1 "alu": {
2   "aluop_size": 2,
3   "func_size": 6,
4   "control_size": 3,
5   "control": [
6     {"aluop": 0, "out": {"control": 2}},
7     {"aluop": 2, "func": 32, "out": {"control": 2}},
```

```
8       ...
9     ],
10    "operations": {
11      "2": "add",
12      ...
13    }
14  }
```

As propriedades `aluop_size`, `func_size` e `control_size` definem os tamanhos (em bits) das entradas `ALUOp` e `func` do controlador da ALU e da entrada de controlo da ALU, respectivamente.

A subsecção `control` define como o controlador da ALU funciona (num *array* JSON). Cada elemento no *array* define (como um objecto JSON) os valores das saídas para os valores das entradas especificados. As propriedades `aluop` e `func` correspondem aos valores das entradas para a correspondência. Se o valor da entrada `func` não for relevante para a correspondência (*don't care*, pode ser qualquer valor), a propriedade pode ser omitida. A propriedade `out` define os valores das saídas para a correspondência. Os valores são especificados como um objecto JSON e os tamanhos (em bits) das saídas são determinados automaticamente. Todos os valores são em formato decimal.

A subsecção `operations` define (como um objecto JSON) a correspondência entre os valores da entrada de controlo da ALU e a operação aritmética que a mesma realiza. As operações disponíveis são: **add**, **sub**, **and**, **or**, **slt**, **xor**, **sll**, **srl**, **sra**, **nor**, **mult**, **div**, **mfhi**, **mflo**. Note que as últimas quatro operações exigem uma ALU "estendida" no CPU, ao invés de uma ALU "normal".