

CA#2 Report

آپکود ها استاندارد هستند به جز jr که 000001 است .

• مسیر داده :

ساختار کلی مسیر داده همانند مسیر داده نشان داده شده در درس است و دستورات گفته در درس همانطور پیاده سازی شده اند. برای pc بعدی یه mux چهار ورودی قرار داده شده که به جز ورودی $pc + 4$ ، یک ورودی آن مستقیم از read Data 1 رجیستر برای دستور jr می آید. ورودی دیگر آن برای دستور j / jal است که 4 بیت اول pc را به اول آدرس آن و 2 صفر به آخر آن اضافه شده است و ورودی آخر برای دستور beq / bne است که آفست داده شده در دستور با $pc + 4$ جمع میشود و وارد آن میشود. این mux با سیگنال pc_src کنترل میشود. همینطور برای mux پشت write_register نیز یک ورودی عدد ثابت 31 برای دستور jal در نظر گرفته شده است. به mux ورودی write_data رجیستر هم یک ورودی اضافه شده که $pc + 4$ در آن قرار میگیرد. این ورودی برای دستور jal است تا آدرس دستور بعدی را در رجیستر 31 ذخیره کرد.

• کنترلر :

کنترلر با توجه به آپکود (6 بیت اول دستور) سیگنال های مورد نیاز را ارسال میکند. کنترلر یک زیرماژول هم برای سیگنال مورد نیاز ALU دارد که در زیرماژول ها توضیح داده شده است. سیگنال zero از ALU نیز وارد کنترلر میشود تا در تصمیم گیری pc_src برای دستورهای beq / bne استفاده شود. 6 بیت آخر هم به عنوان func وارد کنترلر میشود تا در صورتی که دستور از نوع R-Type بود در زیرماژول alu_controller استفاده شود. سیگنال ها نیز در جدول نشان داده شده اند.

• زیرماژول ها :

- InstructionMem : از reg دو بعدی برای ذخیره دستور ها استفاده شده (512 تا 32 reg بیتی) . دستور ها از فایل instructions.data خوانده میشوند و در این ساختمان داده ریخته میشوند. ورودی ماژول آدرس 32 بیتی است که با تقسیم به 4 کردن آن، ایندکس در آرایه به درست می آید و آن را در خروجی قرار میدهد.

- **DataMem** : شیوه نگه داره و پر کردن داده و ایندکسینگ همانند InstructionMem است و از فایل **memory.data** خوانده میشود. با خوردن کلاک در صورت بودن سیگنال **mem_write** ورودی **write_data** در آدرس ریخته میشود.

- در صورت وجود سیگنال **mem_read** نیز داده موجود در آدرس بر روی خروجی **read_data** قرار میگیرد.
- **RegFile** : از **reg** دو بعدی (32 تا 32 بیتی) برای نگه داری استفاده شده است. ایندکسینگ در این مازول همان عدد ورودی است. با خوردن کلاک در صورت وجود سیگنال **reg_write** ، دیتای **write_data** در رجیستر شماره **write_reg_address** ریخته میشود.
- خروجی های **read_data** نیز همیشه محتویات آدرس های **read_reg1/2** را نشان میدهند.
- مقدار **R0** نیز همیشه صفر باقی میماند.

- **PC** : خروجی آن نشان دهنده آدرس دستور بعدی است. در صورت وجود سیگنال **rst** مقدار آن صفر میشود و با هر کلاک نیز مقدار **next_pc** را در خروجی خود قرار میدهد.

- **ALU** : عمل مورد نظر که با **alu_op** مشخص شده و از **ALU_controller** دریافت میشود را روی دو ورودی 32 بیتی خود اعمال میکند و در خروجی قرار میدهد. همینطور در صورت صفر بودن خروجی سیگنال **zero** را به بیرون میدهد تا در کنترلر استفاده شود. عمل های **alu** در جدول مشخص شده است.

- **ALU_Controller** : با دریافت 6 بیت **func** و دو بیت **alu_case** عمل موردنیاز **ALU** را مشخص میکند. حالت های آن در جدول آمده است.

- **Shift2** : ورودی خود را دو بیت به سمت چپ شیفت میدهد.

- **SignExtend** : ورودی 16 بیت میگیرد و با رعایت علامت آن را به 32 بیت تبدیل میکند. (برای استفاده آفست دستور **beq/bne** در **adder**)

- **Adder** : دو ورودی 32 بیتی خود را جمع میکند و در خروجی قرار میدهد.

- **MUX** : چهار نوع **mux** استفاده شده است.

1. 4 ورودی 32 بیت برای ورودی **PC**

2. 2 ورودی 32 بیت برای ورودی دوم **ALU**

3.3 ورودی 5 بیتی برای ورودی آدرس write_reg

3.4 ورودی 32 بیتی برای ورودی write_data رجیستر

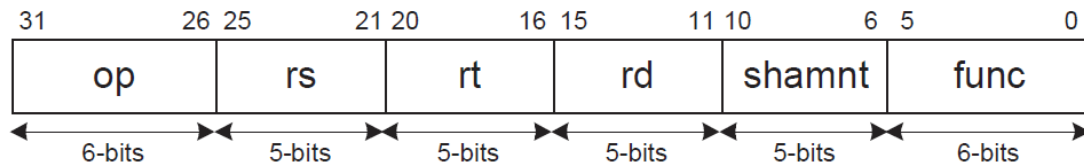
مسیر داده و کنترلر در ماژول Mips32 به هم متصل شده اند .

برای تست برنامه :

می توانید نام فایل ها را در ماژول های InstructionMem و DataMem که در حال حاضر instruction.data و memory.data هستند و به صورت دیفالت محتوای برنامه ی اول را دارد ، به instruction_Q2.data و memory_Q2.data تغییر بدهید و محتوای برنامه ی دوم را چک کنید .
یا میتوانید نام فایل ها را در ماژول ها را ثابت نگه دارید ، و نام اصلی فایل های برنامه ی دوم یا تست خودتان را به instruction.data و memory.data تغییر بدهید و برنامه را تست کنید .

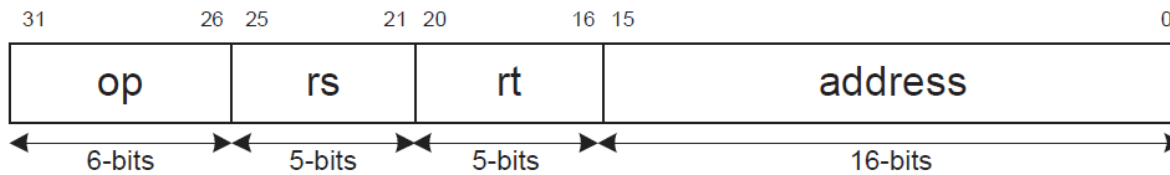
R type format :

➔ add-sub-and-or-slt-jr



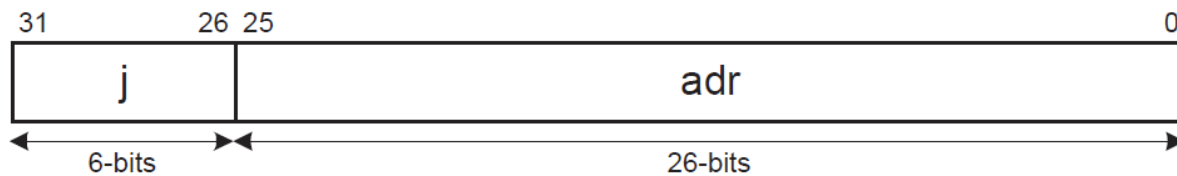
I type format :

➔ lw-sw-beq-bne-addi-andi



J type format :

➔ j-jal



Control Signals

| Instruction | Type | Opcode | Func | Meaning | ALU Op | ALU Src | Reg Dst | Reg Write | Mem To Reg | Mem Read | Mem Write | PC src |
|-------------|------|--------------------|--------|-----------------------------|--------|---------|---------|-----------|------------|----------|-----------|--------|
| add | R | 000000 | 100000 | Add | 10 | 0 | 01 | 1 | 0 | 0 | 0 | 00 |
| Sub | R | 000000 | 100010 | Subtract | 10 | 0 | 01 | 1 | 0 | 0 | 0 | 00 |
| And | R | 000000 | 100100 | Bitwise AND | 10 | 0 | 01 | 1 | 0 | 0 | 0 | 00 |
| Or | R | 000000 | 100101 | Bitwise OR | 10 | 0 | 01 | 1 | 0 | 0 | 0 | 00 |
| Slt | R | 000000 | 101010 | Set to 1 if Less Than | 10 | 0 | 01 | 1 | 0 | 0 | 0 | 00 |
| Jr | R | 000001 (000000) | 001000 | Jump to Address in Register | - | - | - | - | - | - | - | 01 |
| Lw | I | 100011 | NA | Load Word | 00 | 1 | 00 | 1 | 1 | 1 | 0 | 00 |
| Sw | I | 101011 | NA | Store Word | 00 | 1 | - | 0 | - | 0 | 1 | 00 |
| Addi | I | 001000 | NA | Add Immediate | 00 | 1 | 00 | 1 | 0 | 0 | 0 | 00 |
| Andi | I | 001100 | NA | Bitwise AND Immediate | 11 | 1 | 00 | 1 | 0 | 0 | 0 | 00 |
| Beq | I | 000100 | NA | Branch if Equal | 01 | 1 | - | 0 | - | - | - | 00/11 |
| Bne | I | 000101 | NA | Branch if Not Equal | 01 | 1 | - | 0 | - | - | - | 00/11 |
| J | J | 000010 | NA | Jump to Address | - | - | - | - | - | - | - | 10 |
| Jal | J | 000011 | NA | Jump and Link | - | - | 10 | - | 10 | - | - | 10 |

mem to reg sel

| | |
|----|------------|
| 00 | ALU result |
| 01 | Read data |
| 10 | PC + 4 |

reg dst sel

| | |
|----|------------------------|
| 00 | [20:16] instruction |
| 01 | [15:11] instruction |
| 10 | 32'b32 |

PC src sel

| | |
|----|-----------|
| 00 | PC + 4 |
| 01 | jr |
| 10 | j - jal |
| 11 | beq - bne |

ALU OPERATION

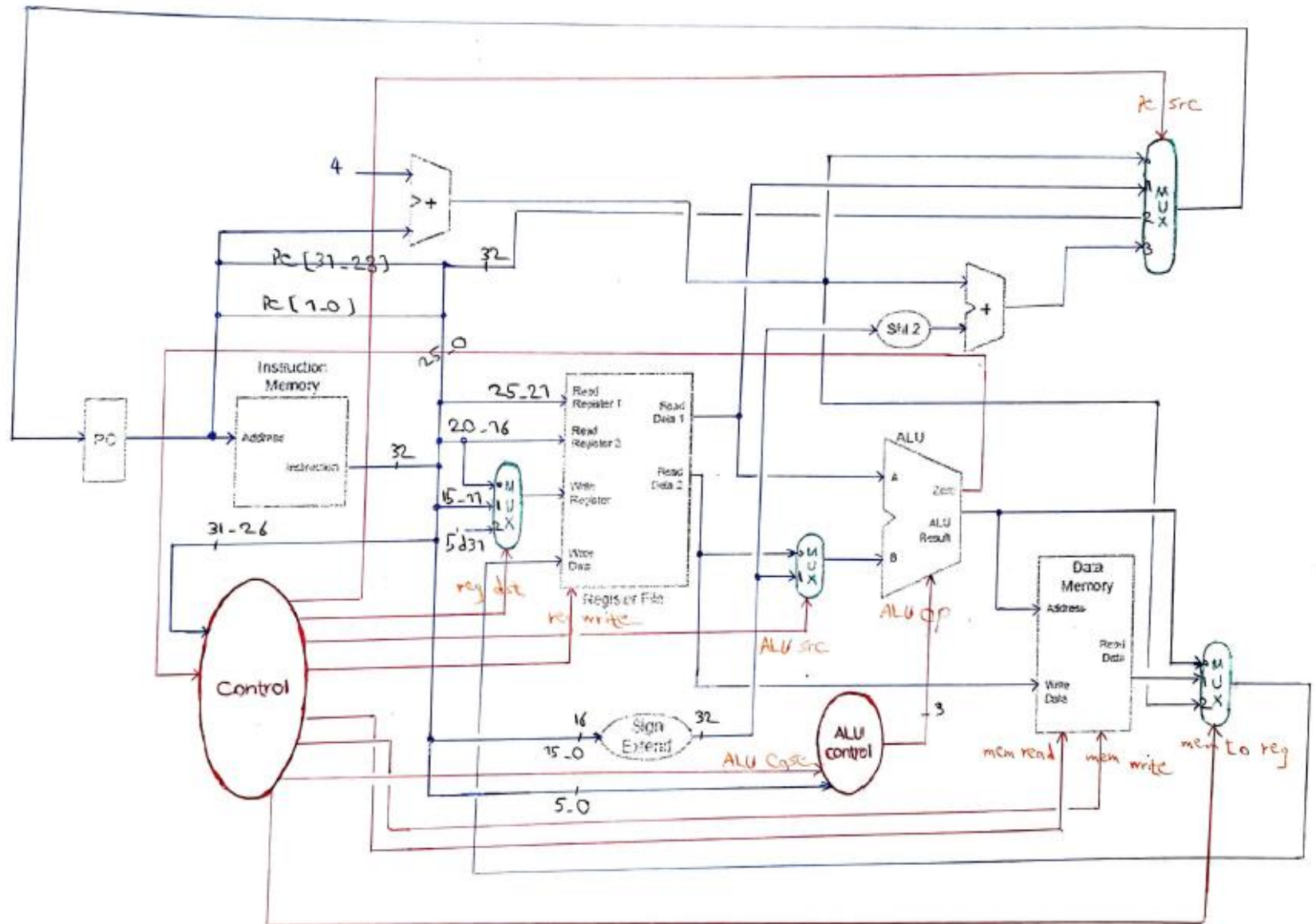
| | |
|-----|-----|
| 000 | and |
| 001 | or |
| 010 | add |
| 110 | sub |
| 111 | slt |

ALU case

| | |
|----|--------------------|
| 00 | lw - sw - addi |
| 01 | Beq-bne |
| 10 | and-or-add-sub-slt |
| 11 | andi |

ALU src sel

| | |
|---|-------------|
| 0 | read data 2 |
| 1 | address |



برنامه اول : دستورات در اینستراکشن و آرایه اعداد در مموری

- $110 + 5 + 0 + (-10) + 1000 + (-2) + 53 + 41 + (-1000) + 32 = 229$

```

F instructions.data X  F memory.data
C: > altera > 13.0sp1 > CA_ca2_test > F instructions.data
1      001000_00000_00001_00000000000000000000 // addi R1, R0, 0 #loop_counter
2      001000_00000_00010_00000000000000001010 // addi R2, R0, 10 #loop_end
3      001000_00000_00011_00000000000000000000 // addi R3, R0, 0 #address_counter
4      001000_00000_00100_00000000000000000000 // addi R4, R0, 0 #sum
5      000100_00001_00010_0000000000000000101 // LOOP : beq R1, R2, END_LOOP
6      100011_00011_00101_0000001111101000 // lw R5, 1000(R3)
7      000000_00100_00101_00100_00000_100000 // addi R4, R4, R6
8      001000_00001_00001_00000000000000000001 // addi R1, R1, 1
9      001000_00011_00011_00000000000000000100 // addi R3, R3, 4
10     000010_0000000000000000000000000000100 // j LOOP
11     101011_00000_00100_0000011111010000 // END_LOOP : sw R4, 2000(R0)
12     00000000000000000000000000000000000000
13     00000000000000000000000000000000000000
14     00000000000000000000000000000000000000

```

```

F instructions.data  F memory.data X
C: > altera > 13.0sp1 > CA_ca2_test > F memory.data
250     00000000000000000000000000000000000000
251     000000000000000000000000000000001101110 //110
252     0000000000000000000000000000000000000101 //5
253     0000000000000000000000000000000000000000 //0
254     1111111111111111111111111111111110110 // -10
255     000000000000000000000000000000001111101000 //1000
256     1111111111111111111111111111111111111110 // -2
257     00000000000000000000000000000000000000110101 //53
258     00000000000000000000000000000000000000101001 //41
259     111111111111111111111111111110000011000 // -1000
260     00000000000000000000000000000000000000100000 //32
261     00000000000000000000000000000000000000000000
262     00000000000000000000000000000000000000000000

```


در عکس اول محتوای رجیستر های استفاده شده در دستورات و در عکس دوم محتوای حافظه (آرایه ی اعداد با شروع از خانه ی 1000) و عکس سوم محتوای خانه ی 2000 حافظه و نتیجه ی نهایی برنامه :

| /TB/mips/dp/regfile/reg_memory | | |
|--------------------------------|--------------|--------------|
| [0] | 32'h00000000 | 32'h00000000 |
| [1] | 32'h00000000 | 32'h00000000 |
| [2] | 32'h0000000a | 32'h0000000a |
| [3] | 32'h0000000a | 32'h0000000a |
| [4] | 32'h00000028 | 32'h00000028 |
| [5] | 32'h000000e5 | 32'h000000e5 |
| [6] | 32'h00000020 | 32'h00000020 |
| [7] | 32'h00000000 | 32'h00000000 |

| | | Msgs |
|-------|---------------|---------------|
| [250] | 32'h0000006e | 32'h0000006e |
| [251] | 32'h00000005 | 32'h00000005 |
| [252] | 32'h00000000 | 32'h00000000 |
| [253] | 32'hffffff6 | 32'hffffff6 |
| [254] | 32'h000003e8 | 32'h000003e8 |
| [255] | 32'hfffffffe | 32'hfffffffe |
| [256] | 32'h00000035 | 32'h00000035 |
| [257] | 32'h00000029 | 32'h00000029 |
| [258] | 32'hffffffc18 | 32'hffffffc18 |
| [259] | 32'h00000020 | 32'h00000020 |
| [260] | 32'h00000000 | 32'h00000000 |
| [261] | 32'h00000000 | 32'h00000000 |

| | | |
|-------|--------------|--------------|
| [498] | 32'h00000000 | 32'h00000000 |
| [499] | 32'h00000000 | 32'h00000000 |
| [500] | 32'h000000e5 | 32'h000000e5 |
| [501] | 32'h00000000 | 32'h00000000 |
| [502] | 32'h00000000 | 32'h00000000 |
| [503] | 32'h00000000 | 32'h00000000 |
| [504] | 32'h00000000 | 32'h00000000 |

برنامه دوم :

- MUX = 110 and index of MUX = 17

[illegible]












































```













instructions.data × memory.data
C: > altera > 13.0sp1 > CA_ca2_test > instructions.data
1 001000_00000_00001_000000000000000000 // addi R1, R0, 0 #loop_counter
2 001000_00000_00010_00000000000010100 // addi R2, R0, 20 #loop_end
3 001000_00000_00011_00000000000000000 // addi R3, R0, 0 #address_counter
4 001000_00000_00100_00000000000000000 // addi R4, R0, 0 #max
5 001000_00000_00101_00000000000000000 // addi R5, R0, 0 #max_index
6 000100_00001_00010_0000000000001001 // Loop : beq R1, R2, END_LOOP
7 100011_00011_00110_0000001111101000 // lw R6, 1000(R3) #data
8 000000_00100_00110_00111_00000_101010 // slt R7, R4, R6 #max < current
9 000100_00111_00000_0000000000000010 // bqe R7, R0, END_DO_UPDATE
10 000000_00000_00110_00100_00000_100000 // DO_UPDATE : add R4, R0, R6
11 000000_00000_00001_00101_00000_100000 // add R5, R0, R1
12 00000000000000000000000000000000 // END_DO_UPDATE
13 001000_00001_00001_00000000000000001 // addi R1, R1, 1
14 001000_00011_00011_00000000000000100 // addi R3, R3, 4
15 000010_000000000000000000000000101 // j LOOP
16 101011_00000_00100_0000011111010000 // END_LOOP : sw R4, 2000(R0)
17 101011_00000_00101_0000011111010100 // sw R5, 2004(R0)
18 00000000000000000000000000000000

```

در عکس اول محتوای رجیسترهای استفاده شده در دستورات و در عکس دوم محتوای حافظه (آرایه ی اعداد با شروع از خانه ی 1000) و عکس سوم محتوای خانه ی 2000 و 2004 حافظه و نتیجه ی نهایی برنامه :

| /TB/mips/dp/regfile/reg_memory | | 32'h00000000 32... | 32'h00000000 32'h0 |
|--------------------------------|--|--------------------|--------------------|
| [0] | | 32'h00000000 | 32'h00000000 |
| [1] | | 32'h00000014 | 32'h00000014 |
| [2] | | 32'h00000014 | 32'h00000014 |
| [3] | | 32'h00000050 | 32'h00000050 |
| [4] | | 32'h0000006e | 32'h0000006e |
| [5] | | 32'h00000011 | 32'h00000011 |
| [6] | | 32'hfffffffcd | 32'hfffffffcd |
| [7] | | 32'h00000000 | 32'h00000000 |
| [8] | | 32'h00000000 | 32'h00000000 |

| | | |
|---|--------------|--------------|
|   [250] | 32'h0000000f | 32'h0000000f |
|   [251] | 32'h00000020 | 32'h00000020 |
|   [252] | 32'h00000035 | 32'h00000035 |
|   [253] | 32'h00000029 | 32'h00000029 |
|   [254] | 32'hffffffc3 | 32'hffffffc3 |
|   [255] | 32'h00000000 | 32'h00000000 |
|   [256] | 32'h00000053 | 32'h00000053 |
|   [257] | 32'hffffffa5 | 32'hffffffa5 |
|   [258] | 32'h0000000b | 32'h0000000b |
|   [259] | 32'h00000017 | 32'h00000017 |
|   [260] | 32'h0000001d | 32'h0000001d |
|   [261] | 32'hffffff6 | 32'hffffff6 |
|   [262] | 32'h00000012 | 32'h00000012 |
|   [263] | 32'h00000048 | 32'h00000048 |
|   [264] | 32'h00000014 | 32'h00000014 |
|   [265] | 32'h00000013 | 32'h00000013 |
|   [266] | 32'hffffff88 | 32'hffffff88 |
|   [267] | 32'h0000006e | 32'h0000006e |
|   [268] | 32'hffffff92 | 32'hffffff92 |
|   [269] | 32'hffffffcd | 32'hffffffcd |
|   [270] | 32'h00000000 | 32'h00000000 |

| | | |
|---|--------------|--------------|
|   [499] | 32'h00000000 | 32'h00000000 |
|   [500] | 32'd110 | 32'd110 |
|   [501] | 32'd17 | 32'd17 |
|   [502] | 32'h00000000 | 32'h00000000 |
|   [503] | 32'h00000000 | 32'h00000000 |
|   [504] | 32'h00000000 | 32'h00000000 |