The Virtual Embedded Architectures Project

0×1 Architecture
0×1 Instruction Set
About the project
The Mini-A-Team
Coming soon

0×1 Instruction Set

Opcode table

Arithmetic	Logic	Data	Control	
0×00 ADD	0×09 AND	0×10 LOAD	0×16 BRAE	0×17 BRANE
0×01 SUB	0×0A OR	0×11 STOR	0×18 BRAL	0×19 BRALE
0×02 MULT	0×0B EXOR	0×12 RLOAD	0×1A BRAG	0×1B BRAGE
0×03 DIV	0×0C SHL	0×13 RSTOR	0×1C INT	0×1D IRET
0×04 MOD	0×0D SHR			

Pseudoinstructions

Arithmetic	Logic	Data	Control	
NEG	INV	MOVR	BRA	NOP
		MOVI	FLUSH	

Arithmetic instructions

ADD - Addition

General usage:	**ADD rA, rB, (rC + imm32)**
Operation Detail:	reg (rA) = reg (rB) + (reg (rC) + imm32);
Opcode:	0×00
Example Usage:	ADD r6, r4, (104 + r5)
Example Encoding (bin):	00000 ??P 00000110 00000100 00000101 00000000_00000000_00000000_01101000
Example Encoding (hex):	0×0006040500000068

SUB - Subtraction

General usage:	**SUB rA, rB, (rC + imm32)**
Operation Detail:	reg (rA) = reg (rB) - (reg (rC) + imm32);
Opcode:	0×01
Example Usage:	SUB r3, r8, (15 + r5)
Example Encoding (bin):	00001 ??P 00000011 00001000 00000101 00000000_00000000_00000000_00001111
Example Encoding (hex):	0×0803080500000000F

MULT - Multiplication

General usage:	**MULT rA, rB, (rC + imm32)**
Operation Detail:	reg (rA) = reg (rB) * (reg (rC) + imm32);
Opcode:	0×02
Example Usage:	MULT r13, r7, (30 + r2)
Example Encoding (bin):	00010 ??P 00001101 00000111 00000010 00000000_00000000_00000000_00011110
Example Encoding (hex):	0×100D07020000001E

DIV - Division to find quotient

General usage:	**DIV rA, rB, (rC + imm32)**
Operation Detail:	reg (rA) = reg (rB) / (reg (rC) + imm32);
Opcode:	0×03
Example Usage:	DIV r8, r5, (14 + r7)
Example Encoding (bin):	00011 ??P 00001000 00000101 00000111 00000000_00000000_00000000_00001110
Example Encoding (hex):	0×180805070000000E

MOD - Modulus division to find remainder

General usage:	**MOD rA, rB, (rC + imm32)**
Operation Detail:	reg (rA) = reg (rB) % (reg (rC) + imm32);
Opcode:	0×04
Example Usage:	MOD r10, r20, (17 + r30)
Example Encoding (bin):	00100 ??P 00001010 00010100 00011110 00000000_00000000_00000000_00010001
Example Encoding (hex):	0×200A141E00000011

Logic instructions

AND - Bitwise AND

General usage:	**AND rA, rB, (rC + imm32)**
Operation Detail:	reg (rA) = reg (rB) & (reg (rC) + imm32);
Opcode:	0×09
Example Usage:	AND r2, r3, (0×98765432 + r4)
Example Encoding (bin):	01001 ??P 00000010 00000011 00000100 10011000_01110110_01010100_00110010

Example Encoding (hex):	0×4802030498765432

OR - Bitwise OR

General usage:	**OR rA, rB, (rC + imm32)**
Operation Detail:	reg (rA) = reg (rB) | (reg (rC) + imm32);
Opcode:	0×0A
Example Usage:	OR r5, r6, (0xBADCC0DE + r7)
Example Encoding (bin):	01010 ??P 00000101 00000110 00000111 10111010_11011100_11000000_11001110
Example Encoding (hex):	0×50050607BADCC0DE

EXOR - Bitwise Exclusive OR

General usage:	**EXOR rA, rB, (rC + imm32)**
Operation Detail:	reg (rA) = reg (rB) ^ (reg (rC) + imm32);
Opcode:	0×0B
Example Usage:	EXOR r8, r9, (r10 + 0xFEEDCAFE)
Example Encoding (bin):	01011 ??P 00001000 00001001 00001010 11111110_11101101_11001010_11111110
Example Encoding (hex):	0×5808090AFEEDCAFE

SHL - Shift left

General usage:	**SHL rA, rB, (rC + imm32)**
Description:	Logically shifts bits to the left a calculated number of positions, bringing 0 into opened low order bits.
Operation Detail:	reg (rA) = reg (rB) << (reg (rC) + imm32);
Opcode:	0×0C
Example Usage:	SHL r11, r12, (r13 + 0xBA5EBA11)
Example Encoding (bin):	01100 ??P 00001011 00001100 00001101 10111010_01011110_10111010_00010001

Example Encoding (hex):	0×600B0C0DBA5EBA11

SHR - Shift right

General usage:	`SHR rA, rB, (rC + imm32)`
Description:	Logically shifts bits to the right a calculated number of positions, bringing 0 into opened high order bits.
Operation Detail:	`reg (rA) = reg (rB) >> (reg (rC) + imm32);`
Opcode:	0×0D
Example Usage:	`SHR r14, r15, (r1 + 0xB01DFACE)`
Example Encoding (bin):	01101 ??P 00001110 00001111 00000001 10110000_00011101_11111010_11001110
Example Encoding (hex):	0×680E0F01B01DFACE

Data instructions

LOAD - Load a register from memory

General usage:	`LOAD rA, [rC + imm32]`
Description:	Register A is loaded with a copy of the data at the effective address found by adding the contents of register C to an immediate.
Operation Detail:	`reg (rA) = mem (reg (rC) + imm32);`
Opcode:	0×10
Example Usage:	`LOAD r2, [r3 + 4]`
Example Encoding (bin):	10000 ??P 00000010 ???????? 00000011 00000000_00000000_00000000_00000100
Example Encoding (hex):	0×8002000300000004

STORE - Store a register value in memory

General usage:	`STORE [rC + imm32], rA`
Description:	The data in register A is copied to the effective address found by adding the contents of register C to an immediate.
Operation Detail:	`mem (reg (rC) + imm32) = reg (rA);`
Opcode:	0×11
Example Usage:	`STORE [r4 -1], r5`
Example Encoding (bin):	10001 ??P 00000101 ???????? 00000100 11111111_11111111_11111111_11111111
Example Encoding (hex):	0×88050004FFFFFFFF

RLOAD - Load a register from an indexed register

General usage:	`RLOAD rA, [rC + imm32]`
Description:	Register A is loaded with a copy of the data in the register with the effective index found by adding the contents of register C to an immediate.
Operation Detail:	`reg (rA) = reg (reg (rC) + imm32);`
Opcode:	0×12
Example Usage:	`RLOAD r2, [r3 + 4]`
Example Encoding (bin):	10010 ??P 00000010 ???????? 00000011 00000000_00000000_00000000_00000100
Example Encoding (hex):	0×9002000300000004

RSTORE - Store a register value in an indexed register

General usage:	`RSTORE [rC + imm32], rA`
Description:	The data in register A is copied to the register with the effective index found by adding the contents of register C to an immediate.
Operation Detail:	`reg (reg (rC) + imm32) = reg (rA);`
Opcode:	0×13
Example Usage:	`RSTORE [r4 -1], r5`

Example Encoding (bin):	10011 ??P 00000101 ???????? 00000100 11111111_11111111_11111111_11111111
Example Encoding (hex):	0×98050004FFFFFFFF

Control instructions

The following conditional branch instructions provide a hint to the hardware about the likelihood a branch will be taken. A braced value, “{true}” or “{false}”, is given to indicate respectively whether the branch should be prefetched or not. This static hinting approach is obviously not the greatest branch prediction technique, but it will allow students debugging their applications to easily predict what will be fetched to the pipeline. When encoded, the Hint bit will be set to 1 if the hint is “{true}” and cleared to 0 for “{false}”.

BRAE - Branch if equal

General usage:	**BRAE {hint} [rC + imm32], rA, rB**
Description:	The Program Counter (PC) register is replaced by the effective address found by adding the contents of register C to an immediate only if the value in register A is equal to the value in register B.

Operation Detail:	`if (reg (rA) == reg (rB)) {`
	`  reg (PC) = reg (rC) + imm32;`
	`}`

Opcode:	0×16
Example Usage:	BRAE [r2 + 0x1ADA], r3, r4
Example Encoding (bin):	10110 ?HP 00000011 00000100 00000010 00000000_00000000_00011010_11011010
Example Encoding (hex):	0xB003040200001ADA
Notes:	

BRANE - Branch if not equal

General usage:	

General usage: **BRANE {hint} [rC + imm32], rA, rB**

Description: The Program Counter (PC) register is replaced by the effective address found by adding the contents of register C to an immediate only if the value in register A is not equal to the value in register B.

Operation Detail:
```
if (reg (rA) == reg (rB)) {
   reg (PC) = reg (rC) + imm32;
}
```

Opcode: 0×17

Example Usage: `BRANE [r2 + 0x1ADA], r3, r4`

Example Encoding (bin): 10111 ?HP 00000011 00000100 00000010 00000000_00000000_00011010_11011010

Example Encoding (hex): 0xB803040200001ADA

Notes:

BRAL - Branch if less than

General usage: **BRAL {hint} [rC + imm32], rA, rB**

Description: The Program Counter (PC) register is replaced by the effective address found by adding the contents of register C to an immediate only if the value in register A is less than the value in register B.

Operation Detail:
```
if (reg (rA) < reg (rB)) {
   reg (PC) = reg (rC) + imm32;
}
```

Opcode: 0×18

Example Usage: `BRAL [r11 + 0x2000], r12, r13`

Example Encoding (bin): 11000 ?HP 00001100 00001101 00001011 00000000_00000000_00100000_00000000

Example Encoding (hex): 0xC00C0D0B00002000

BRALE - Branch if less than or equal

General usage: **BRALE {hint} [rC + imm32], rA, rB**

Description: The Program Counter (PC) register is replaced by the effective address found by adding the contents of register C to an immediate only if the value in register A is less than or equal to the value in register B.

Operation Detail:
```
if (reg (rA) <= reg (rB)) {
   reg (PC) = reg (rC) + imm32;
}
```

Opcode: 0×19

Example Usage: `BRALE [r14 + 0x1BAA], r15, r1`

Example Encoding (bin): 11001 ?HP 00001111 00000001 00001110 00000000_00000000_00011011_10101010

Example Encoding (hex): 0xC80F010E00001BAA

BRAG – Branch if greater than

General usage: **`BRAG {hint} [rC + imm32], rA, rB`**

Description: The Program Counter (PC) register is replaced by the effective address found by adding the contents of register C to an immediate only if the value in register A is greater than the value in register B.

Operation Detail:
```
if (reg (rA) < reg (rB)) {
   reg (PC) = reg (rC) + imm32;
}
```

Opcode: 0×1A

Example Usage: `BRAG [r11 + 0x2000], r12, r13`

Example Encoding (bin): 11010 ?HP 00001100 00001101 00001011 00000000_00000000_00100000_00000000

Example Encoding (hex): 0xD00C0D0B00002000

BRAGE – Branch if greater than or equal

General usage: **`BRAGE {hint} [rC + imm32], rA, rB`**

Description: The Program Counter (PC) register is replaced by the effective address found by adding the contents of register C to an immediate only if the value in register A is greater than or equal to the value in register B.

Operation Detail:
```
if (reg (rA) <= reg (rB)) {
   reg (PC) = reg (rC) + imm32;
}
```

Opcode: 0×1B

Example Usage: `BRAGE [r14 + 0x1BAA], r15, r1`

Example Encoding (bin):	11011 ?HP 00001111 00000001 00001110 00000000_00000000_00011011_10101010
Example Encoding (hex):	0xD80F010E00001BAA

INT – Generate an interrupt

General usage:	**INT (rC + imm32)**
Description:	Sets the Interrupt Flag Register bit for the interrupt number found by adding register C to an immediate.
Operation Detail:	`IFR = IFR | (1 << (reg (rC) + imm32));`
Opcode:	0×1C
Example Usage:	`INT (r9 + 1)`
Example Encoding (bin):	11100 ??P ???????? ???????? 00001001 00000000_00000000_00000000_00000001
Example Encoding (hex):	0xE000000900000001

IRET – Return from an interrupt

General usage:	**IRET**
Description:	Soon to come… the complete details of the mystifying and fabulous IVBT interrupt handler!
Opcode:	0×1D
Example Usage:	`IRET`
Example Encoding (bin):	11101 ??P ???????? ???????? ???????? ????????_????????_????????_????????
Example Encoding (hex):	0xE800000000000000

Pseudoinstructions

The following pseudo instructions are defined soley to make programs more readable. However, these pseudo instructions are each replaced with a single core instruction; ***NOT*** a sequence of instructions.

NEG - Negation pseudoinstruction

General usage:	**NEG rA, rC**
Description:	Negates the value of register C, storing the result into register register A.
Operation Detail:	SUB rA, r0, (rC + 0)
Example Usage:	NEG r4, r5
Example Encoding (bin):	00001 ??P 00000100 00000000 00000101 00000000_00000000_00000000_00000000
Example Encoding (hex):	0×0804000500000000

INVERT - Inverts a register pseudoinstruction

General usage:	**INV rA, rB**
Description:	Invert all bits of the value contained in register B, storing the result into register A.
Operation Detail:	EXOR rA, rB, (r0 + 0xFFFFFFFF)
Example Usage:	INV r7, r11
Example Encoding (bin):	01011 ??P 00000111 00001011 00000000 11111111_11111111_11111111_11111111
Example Encoding (hex):	0×58070B00FFFFFFFF

MOVR - Move register pseudoinstruction

General usage:	**MOV rA, rB**
Description:	Copies the contents of register B into register A.
Operation Detail:	ADD rA, rB, (r0 + 0)
Example Usage:	MOV r5, r6
Example Encoding (bin):	00000 ??P 00000101 00000110 00000000 00000000_00000000_00000000_00000000
Example Encoding (hex):	0×0005060000000000

MOVI - Move immediate pseudoinstruction

General usage:	**MOV rA, imm32**
Description:	Copies an immediate into register A.
Operation Detail:	ADD rA, r0, (r0 + imm32)
Example Usage:	MOV r3, 17
Example Encoding (bin):	00000 ??P 00000011 00000000 00000000 00000000_00000000_00000000_00010001
Example Encoding (hex):	0×0003000000000011

BRA - Unconditional branch pseudoinstruction

General usage:	**BRA [rC + imm32]**
Description:	The Program Counter (PC) register is replaced by the effective address found by adding the contents of register C to the immediate value.
Operation Detail:	BRAE {T} [rC + imm32], r0, r0
Example Usage:	BRA [r2+0x2300]
Example Encoding (bin):	10000 ?1P 00000000 00000000 00000010 00000010_00000011_00000000_00000000
Example Encoding (hex):	0×8200000202030000

NOP - No operation performed pseudoinstruction

General usage:	**NOP**
Description:	Exhaust a clock cycle.
Operation Detail:	ADD r0, r0, (r0 + 0)
Example Usage:	NOP
Example Encoding (bin):	00000 ??P 00000000 00000000 00000000 00000000_00000000_00000000_00000000
Example Encoding (hex):	0×0000000000000000

FLUSH – Empty the pipeline pseudoinstruction

General usage:	**FLUSH**
Description:	All pipeline stages are filled with NOPs. The PC queue is emptied, as well, and the address of the instruction scheduled after the flush is added back to the PC queue.
Operation Detail:	`BRAE {F} [PC + 0], r0, r0`
Example Usage:	`FLUSH`
Example Encoding (bin):	`10000 ?0P 00000000 00000000 11111111 00000000_00000000_00000000_00000000`
Example Encoding (hex):	0×800000FF00000000

"Even this could be a MiniAT peripheral."