# Goals

The goal is to come up with a 32-bit embedded controller style ISA. The ISA should be relatively simple to implement, yet powerful enough to support modern operating systems.

# Programming Model

## General Purpose Register Array

The ISA supports 32, 32 bit general purpose integer registers.

|  |  |  |  |
| --- | --- | --- | --- |
| # | Mne. | Usage | Saver |
| 0 | R0 | Always zero |  |
| 1 | R1 | first return value | Caller |
| 2 | R2 | second return value | Caller |
| 3 | R3 | subroutine arguments | Caller |
| 4 | R4 | Caller |
| 5 | R5 | Caller |
| 6 | R6 | Caller |
| 7 | R7 | Caller |
| 8 | R8 | Caller |
| 9 | R9 | Caller |
| 10 | R10 | Caller |
| 11 | R11 | Saved registers | Callee |
| 12 | R12 | Callee |
| 13 | R13 | Callee |
| 14 | R14 | Callee |
| 15 | R15 | Callee |
| 16 | R16 | Callee |
| 17 | R17 | Callee |
| 18 | R18 | Callee |
| 19 | R19 | Callee |
| 20 | R20 | Callee |
| 21 | R21 | Callee |
| 22 | R22 | Temporaries | Caller |
| 23 | R23 | Caller |
| 24 | R24 | Caller |
| 25 | R25 | Caller |
| 26 | TR | task register |  |
| 27 | TP | thread pointer |  |
| 28 | GP | global pointer |  |
| 29 | BP | base / frame pointer |  |
| 30 | SP | stack pointer |  |
| 31 | PC | Program counter alias |  |

The stack pointer register (R30) is automatically updated by instructions that use the stack (JSR, RTS, BRK, RTI, PUSH, and POP).

## Program Counter

The program counter is 23 bits in size. The two LSB’s of the counter are always zero. This allows up to 8MB of sequential code. When combined with the code segment base register 27 bit code addressing is possible.

|  |  |
| --- | --- |
| 22 2 | 10 |
| Offset22..2 | 00 |

The program counter normally just increments by four (which is why it is called a counter), however this behaviour may be altered by a JMP, JSR, RTS, BRK, RTI or branch instruction. Refer to the instruction descriptions to see how these instructions modify the program counter.

The program counter is readable as general purpose register #31 for use in forming program counter relative addresses.

## Segment Selector Registers

The ISA supports eight ten bit segment selector registers.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SPR | SEG | 31 10 | 9 0 |  |
| 18 | 0 |  | DS10 | data selector |
| 19 | 1 |  | ES10 |  |
| 1A | 2 |  | FS10 |  |
| 1B | 3 |  | GS10 |  |
| 1C | 4 |  |  |  |
| 1D | 5 |  |  |  |
| 1E | 6 |  | SS10 | stack selector |
| 1F | 7 |  | CS10 | code selector |

The segment selector register used for address formation is selected by a bit field encoded in the instruction. The assembler will setup a default selector corresponding to the general purpose register used. For instance if the instruction uses the stack pointer, the assembler will automatically select the stack selector as the segment register for the instruction. This default setup may be overridden using a selector prefix in assembler code. The prefix does not output any additional opcodes, it just sets the segment selector register number in the current instruction.

### Modifying

The selector registers are accessed via the MTSPR and MFSPR instructions. It is not possible to modify the code selector by directly moving a value to it via MTSPR. The code segment selector register must be modified using one of the control flow instructions that may affect it (JMP, JSR, RTS, RTI, or BRK).

### Initial Values

Except for the code segment, the segment registers contain undefined values on reset. On reset the code segment base register is set to $FFFF.

# Memory Management

## The Descriptor Table

The descriptor table contains information on the location and size for segments. This table is a special 1k word dual-ported memory. Entries in the table have the following format:

|  |  |  |
| --- | --- | --- |
|  | 31 16 | 15 0 |
| w0 | Limit16 | Base16 |
| w1 | Limit16 | Base16 |
| … |  |  |

The descriptor table is located at a fixed address of $4000\_0000 placing it in the first 4kB of the high memory area.

10 bit selectors are used to index into the table in order to determine the characteristics of the segment.

## Program Code Addressing

The ISA supports a 27 bit code address space. This was chosen for the FPGA board which has 128MB (27 address bits) of memory onboard. A sixteen bit segment base value is shifted left eleven bits and combined with a twenty-three bit offset value to form a twenty-seven bit address. The segment paragraph size is 2kB.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | |  |
| 09 | Offset22.2 | | 00 |
|  | + | |  |
| Base15.0 | | 000010..0 | |
|  | = | |  |
| Address26..0 | | | |

The JSR instruction pushes both the code segment selector and the program counter on the stack. Both of these values can be pushed onto the stack in a single word.

|  |  |  |
| --- | --- | --- |
| 31 | 30 21 | 20 0 |
| 0 | Selector9..0 | Offset22..2 |

The RTS instruction always restored both the code segment selector and the offset.

The JMP and JSR instruction transfer control to an address within the current segment. A 23 bit address range is supported by these instructions, allowing code of up to 8MB in size within a single segment. In order to transfer to code in another segment the JSP (jump segment prefix) instruction is issued just prior to a JMP or JSR.

## Data Memory Addressing

The ISA supports a 27 bit data addressing range (this is the amount of memory available on the FPGA board in use). A sixteen bit segment base value is combined with up to 26 bits of displacement from a load or store instruction to form a final 27 bit address.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | |  |
| 01 | Offset25..0 | | | |
|  | + | | |  |
| Base15.0 | | | 00010..0 | |
|  | | = | |  |
| Address26..0 | | | | |

## High Memory Area

When the offset plus the segment base exceeds 27 bit addressing, the high memory area is active. For instance, by setting the code segment value to a maximum ($FFE0) the processor may reach into the high memory area up to 8MB-64kB. This may be a useful storage place for ROM and IO, leaving a full 128MB for other memory requirements. In fact, on reset the code segment base is set to $FFFF so it naturally points to the last 2kB paragraph of memory.

# Interrupts

An interrupt causes the code selector and program counter to be saved in the return address format on the stack. Next the code segment selector and program counter are loaded from the interrupt vector table. These two items are stored in the vector table in the same format as a return address.

|  |  |  |
| --- | --- | --- |
| 31 | 30 21 | 20 0 |
| 0 | Selector9..0 | Offset22..2 |

The interrupt mask is stored in a backup register, then further interrupts are masked from happening. The processor mode is stored in a backup register then the processor is switched to kernel mode.

## Interrupt Vector Table

The interrupt vector table stores the addresses of interrupt routines. The location of the interrupt vector table is specified by the vector table base address register (VBR).

# Instruction Set Summary

## JSP – Jump Segment Prefix

### Instruction Format:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 26 |  | 20 11 | 10 6 | 5 0 |  |
| 3C6 | ~5 | Imm9..0 | Ra5 | 026 | JSP |

### Description:

Either the immediate value or the contents of a register are used to determine the selector value for the following jump or jump to subroutine instruction. Either the immediate value or the register contents should be zero. The two values (immediate and register) are OR’d together to form the selector value. The code segment selector is loaded with the value from this instruction when a following JMP or JSR takes place. Both the program counter and the code segment selector are altered at the same time as an indivisible operation. If a JSP instruction is erroneously issued without a following JMP or JSR the instruction will be treated as a NOP operation and the code segment selector will not be altered.

Interrupts are disabled between the JSP prefix and the following JMP / JSR instruction.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | |  | | | | |  | |  |  |
| ~17 | | | | | | | | | vectno9 | | | 006 | BRK |
| Address22..2 | | | | | | | | | | | Ra5 | Opcode6 | JMP / JSR |
| Displacement22.2 | | | | | | | | | | | ~5 | 166 | BSR |
| Immed16 | | | | | | | | Rt5 | | | Ra5 | Opcode6 | RI format |
| Func6 | | ~5 | | | | Rt5 | | Rb5 | | | Ra5 | 026 | RR format |
| Func6 | | ~5 | | | | Rt5 | | Imm5 | | | Ra5 | 026 | Shift # |
| 386 | | ~2 | b3 | | | Rc5 | | Rb5 | | | Ra5 | 026 | PUSH |
| 396 | | ~5 | | | | Rt5 | | ~5 | | | ~5 | 026 | POP |
| 3C6 | | ~5 | | | | Imm10 | | | | | Ra5 | 026 | JSP |
| 3F6 | | ~5 | | | | ~5 | | ~5 | | | ~5 | 026 | RTI |
| Disp13 | | | | | | | Op3 | Rb5 | | | Ra5 | Opcode6 | Bcc / Bccu |
| Disp13 | | | | | | | Immed8 | | | | Ra5 | Opcode6 | Bcc # |
| Disp13 | | | | | | | Sg3 | Rt5 | | | Ra5 | Opcode6 | Lx |
| Disp13 | | | | | | | Sg3 | Rb5 | | | Ra5 | Opcode6 | Sx |
| Disp6 | Sg3 | | | Sc2 | | Rt5 | | Rb5 | | | Ra5 | Opcode6 | LxX |
| Disp6 | Sg3 | | | Sc2 | | Rc5 | | Rb5 | | | Ra5 | Opcode6 | SxX |
|  | | | | | | | |  | | |  |  |  |
| Immediate26 | | | | | | | | | | | | 3E6 | IMM |

Major Opcode

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x | BRK |  | {RR} |  | ADD# | SUB# | CMP# | MUL# | DIV# | MOD# |  |  | AND# | OR# | EOR# |  |
| 1x | BEQ# | BNE# | {Bcc} | {Bccu} | JMP | JSR | BSR |  | BLT# | BGE# | BLE# | BGT# | BLTU# | BGEU# | BLEU# | BGTU# |
| 2x | LB | LBU | LH | LHU | LW |  | SEQ# | SNE# | LBX | LBUX | LHX | LHUX | LWX | LEAX |  |  |
| 3x | SLT# | SGE# | SLE# | SGT# | SLTU# | SGEU# | SLEU# | SGTU# | SBX | SHX | SWX | SB | SH | SW | IMM |  |

02 Group Func

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x | NAND | NOR | ENOR | ORC | ADD | SUB | CMP | MUL | DIV | MOD |  |  | AND | OR | EOR | ANDC |
| 1x |  |  |  |  | ADDU | SUBU | CMPU | MULU | DIVU | MODU |  |  | MTSEG | MFSEG | MTSPR | MFSPR |
| 2x | SLL | ROL | SRL | ROR | SRA |  | SEQ | SNE | SLT | SGE | SLE | SGT | SLTU | SGEU | SLEU | SGTU |
| 3x | SLL # | ROL # | SRL # | ROR # | SRA # |  |  |  | PUSH | POP |  |  | JSP |  | RTS | RTI |

12 Group Op3

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 |
| 0x | BEQ | BNE |  |  | BLT | BGE | BLE | BGT |

13 Group Op3

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 |
| 0x |  |  |  |  | BLTU | BGEU | BLEU | BGTU |