General Purpose Register Array

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| r0 | always zero | r32 |  |
| r1 |  | r33 |  |
| r2 |  | r34 |  |
| r3 |  | r35 |  |
| r4 |  | r36 |  |
| r5 |  | r37 |  |
| r6 |  | r38 |  |
| r7 |  | r39 |  |
| r8 |  | r40 |  |
| r9 |  | r41 |  |
| r10 |  | r42 |  |
| r11 |  | r43 |  |
| r12 |  | r44 |  |
| r13 |  | r45 |  |
| r14 |  | r46 |  |
| r15 |  | r47 |  |
| r16 |  | r48 |  |
| r17 |  | r49 |  |
| r18 |  | r50 |  |
| r19 |  | r51 |  |
| r20 |  | r52 |  |
| r21 |  | r53 |  |
| r22 |  | r54 |  |
| r23 |  | r55 |  |
| r24 |  | r56 | thread pointer |
| r25 |  | r57 | global pointer |
| r26 |  | r58 | frame pointer |
| r27 |  | r59 | stack pointer |
| r28 |  | r60 | catch link address |
| r29 |  | r61 | exception return address |
| r30 |  | r62 | interrupt return address |
| r31 |  | r63 | return address |

Instructions

Instructions use fixed 41 bit formats. Three instructions are bundled to fit three within 128 bits with 5 unused bits left over.

|  |  |  |  |
| --- | --- | --- | --- |
| 127 123 | 122 82 | 81 41 | 40 0 |
| ~5 | Slot2 | Slot1 | Slot0 |

Instruction Addresses

The low order four bits of the instruction address are one of 0,5,or A hex. Note that bits 3,2 of the address are always the same as bits 1,0. 0 represents slot 0, 5 represents slot1 and A represents slot2 within a bundle.

# Segmentation Model

### Segmentation Overview

As part of the memory management portion of a cpu segment registers are often provided. There are usually multiple segment registers in order to support multiple segments which are typically part of a program. Common program segment are: the code segment, the data segment, the uninitialized data segment and the stack segment. There are often other segments as well. 80x88 is famous for its segment registers, but other processors like IBM’s PowerPC also use them as well. Segment registers are a fairly easy to understand , and a low cost memory management approach. The memory address from an instruction is added to a value from a segment register in order to form a final address. The segment register is often shifted left as it is added in order to allow a greater physical memory range than the range directly supported by the architecture. Segment registers allow programs to be written as if they had specific memory addresses available to them, such as starting at location zero, while in reality the actual physical address of the program is much different. Once a design seems to be working well, I tend to add segment registers to the design as a first step at providing memory management features.

### Address Formation:

The virtual address is added to a segment base register in order to form a final address. Note that there is no shift associated with the segment addition in this case.

|  |  |
| --- | --- |
|  | Virtual Address |
|  | + |
| segment base register | |
|  | = |
| Segmented address | |

Note that the segment base register may contain a value larger than that of a general purpose register.

Segmentation is applied only when the CPL is numerically greater than three. (CPL > 3). At higher privilege levels segmentation is not applied, and the processor is in a flat addressing mode where the virtual address and the physical address are the same. Only the low order 64 bits of the address space is available when CPL < 4.

## Number of Registers:

The number of segment registers that are useful seems not to be quantified as closely as the number of general purpose registers. However, four registers was deemed not enough for the 80x86 architecture and two more segment registers were added. Also a couple of additional registers in the 80x86 design were added to support the segmentation architecture and they act a lot like segment registers. These include the task register and the local descriptor table register. So we have about eight segment registers in the 80x86 architecture. PA-RISC has eight space registers that act a lot like segment base registers. The PowerPC processor uses an array of sixteen segment registers. The Finitron 64 bit ISA uses sixteen segment registers. Segments registers are typically initialized to a flat memory model then forgotten about.

## Segment Usage Conventions

All program counter addresses for instruction fetches are formed with the code segment.

The stack segment is used for operations involving the stack.

Load / Store operations use the data segment (DS).

Segment Register Array

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | Assoc. GPR |  |
| 0 | NS | NULL segment |  |  |
| 1 | DS | data segment |  |  |
| 2 | TLS | thread storage | TP |  |
| 3 | BSS | BSS segment |  |  |
| 4 | RS | read only segment |  |  |
| 5 | ES | extra segment |  |  |
| 6 | GS | global storage | GP |  |
| 7 | seg7 |  |  |  |
| 8 | seg8 |  |  |  |
| 9 | seg9 |  |  |  |
| 10 | seg10 |  |  |  |
| 11 | LDT | local descriptor table |  |  |
| 12 | TSS | task state segment | TR |  |
| 13 | VDS | volatile data segment |  |  |
| 14 | SS | Stack segment | SP,BP |  |
| 15 | CS | Code segment | PC |  |
|  |  |  |  |  |

## 

## Composition

Each segment register is an array of four words containing the segment information.

|  |  |  |  |
| --- | --- | --- | --- |
| ACR8 | DPL4 | ~52 | |
| Limit 63..0 | | | |
| ~32 | | | Base 95..64 |
| Base 63..00 | | | |

### Moving Segment Register Values

Segment registers may be only moved to or from general purpose registers. There are no other instructions for manipulating a segment register. MTSEG is used to move a general purpose register to the segment register. MFSEG moves a segment register into a general purpose register.

Note that the only way to change the code segment is by executing an interrupt. An interrupt or the break instruction will cause a transfer into the non-segmented area provided for the operating system. The code segment may then be modified without it impacting the current executing stream of instructions. Unlike some other architectures there are no far call / far return instructions.

Interrupts

An issue that arises with the use of interrupts is the management of the interrupt return address including both the segment and offset. The original address prior to the interrupt has to be stored so that it may be restored after the interrupt is processed. Also, the code segment and offset of the address of the interrupt routine needs to be set. The offset portion of the return address is stored in a general purpose register identified by a BRK instruction. But where is the segment portion of the address stored ? It isn’t stored automatically. The segment potion of the address must be stored manually by software via the MFSEG instruction. This can be gotten away with only by using a non-segmented code address space. When an interrupt occurs the processor vectors into the non-segmented code address space which means that the segment register contents are ignored. The contents of the segment register will still be pointing to the code that got interrupted. If the interrupt routine doesn’t branch to a new code segment, then the code segment doesn’t need to be stored off. When the RTI instruction is executed, the processor will return to the privilege level and program offset present before the interrupt occurred.

The address, including both the segment and offset needs to be established in an automatic fashion as a single operation. Part of the address cannot be changed as the processor would become confused about where to run code. This cannot be done by running software without dedicated specialized instructions. For instance one cannot issue an instruction to modify the code segment like: MTSEG CS.base, address. Unless the code segment were to remain stable throughout the operation. Therefore moving to the code segment is allowed from only an unsegmented address space. Unsegmented address space is available at privilege levels CPL < 4.

On an interrupt how does the processor know which segment to branch to ? Interrupt routine addresses are located in the interrupt vector table. The address is an address in the unsegmented area of the instruction space, hence the value of the code segment register is irrelevant. The vector table is 2kB in size and contains 512, 4 byte entries with the following format:

|  |  |  |  |
| --- | --- | --- | --- |
| 31 30 | 29 2 | 1 | 0 |
| PL2 | Address29..2 | ~ | M |

PL2: the two LSB’s of the new privilege level to set (must be numerically lower or equal to current), the upper two bits are assumed to be zero. This forces the processor to an unsegmented address space on interrupt.

M: 1 = mask further interrupts from happening

The interrupt routine must be located within the lowest 1GB of the address space as there are only 30 bits allowed for the interrupt routine address. This design minimizes the size of the interrupt vector table so that cache space isn’t wasted.

The vector table location is specified by the vector table base register (VBR).

## Control Register Zero (SPR 05 or CR0)

This register contains a bit to enable protected mode.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 63 | 62 9 6 1 | | | | | | | | 0 |
| ~ | ~ |  |  |  |  |  | ~ |  | Pe |

PE: Protected Mode enable: 1 = enabled, 0 = disabled.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  | | | | |  | |  | | |  |  |  |
|  | | ~3 | Vector9 | | | | | Rt6 | | ~6 | | 008 | BRK |  |
| ~9 | | Func6 | | | Rt6 | | | Rb6 | | Ra6 | | 028 | {RR} |  |
| ~9 | | 3F6 | | | 06 | | | ~6 | | 3E6 | | 028 | RTI |  |
| ~9 | | Func6 | | | Rt6 | | | Imm6 | | Ra6 | | 028 | Shifts # |  |
| Immed21 | | | | | | | | Rt6 | | Ra6 | | 048 | ADD # |  |
| Immed21 | | | | | | | | Rt6 | | Ra6 | | 058 | SUB # |  |
| Immed21 | | | | | | | | Rt6 | | Ra6 | | 078 | MUL # |  |
| Immed21 | | | | | | | | Rt6 | | Ra6 | | 088 | DIV # |  |
| Immed21 | | | | | | | | Rt6 | | Ra6 | | 098 | MOD # |  |
| Immed21 | | | | | | | | Rt6 | | Ra6 | | 0C8 | AND # |  |
| Immed21 | | | | | | | | Rt6 | | Ra6 | | 0D8 | OR # |  |
| Immed21 | | | | | | | | Rt6 | | Ra6 | | 0E8 | EOR # |  |
| Immed21 | | | | | | | | Rt6 | | ~6 | | 168 | LD # |  |
| Immed21 | | | | | | | | Rt6 | | Ra6 | | 3x8 | Scc # |  |
| ~5 | Cnd4 | Disp4 | | Offs8 | | | | Rb6 | | Ra6 | | 618 | Bcc |  |
| Immed14..6 | | Disp4 | | Offs8 | | | | Imm5..0 | | Ra6 | | 5x8 | Bcc # |  |
| Disp13 | | | | Offs8 | | | | Rt6 | | ~6 | | 668 | BSR |  |
| Immed13 | | | | ~2 | | Rt6 | | Rb6 | | Ra6 | | 608 | RTS |  |
| Offset21 | | | | | | | | Rt6 | | Ra6 | | 648 | JAL |  |
| Disp21 | | | | | | | | Rt6 | | Ra6 | | 808 | LB |  |
| Disp21 | | | | | | | | Rt6 | | Ra6 | | 818 | LBU |  |
| Disp21 | | | | | | | | Rt6 | | Ra6 | | 828 | LC |  |
| Disp21 | | | | | | | | Rt6 | | Ra6 | | 838 | LCU |  |
| Disp13 | | | | Sc2 | | Rt6 | | Rb6 | | Ra6 | | 888 | LBX |  |
| Disp13 | | | | Sc2 | | Rc6 | | Rb6 | | Ra6 | | A88 | SBX |  |
| ~33 | | | | | | | | | | | | E18 | NOP |  |
| Immediate31..0 | | | | | | | | | | | ~ | FB8 | IMM1 |  |
| Immediate63..32 | | | | | | | | | | | ~ | FC8 | IMM2 |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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 |  |  |  |  | ADDU# | SUBU# | LD# | MULU# | DIVU# | MODU# |  |  |  |  |  |  |
| 2x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3x | SEQ# | SNE# |  |  |  |  |  |  | SGT# | SLE# | SGE# | SLT# | SHI# | SLS# | SHS# | SLO# |
| 4x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5x | BEQ # | BNE # |  |  |  |  | BRA # | BRN # | BGT # | BLE # | BGE # | BLT # | BGTU# | BLEU # | BGEU # | BLTU # |
| 6x | RTS | {Brr} |  |  | JAL |  | BSR |  |  |  |  |  |  |  |  |  |
| 7x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 8x | LB | LBU | LC | LCU | LH | LHU | LW | LEA | LBX | LBUX | LCX | LCUX | LHX | LHUX | LW | LEAX |
| 9x | LFS | LFD | LFQ |  |  |  |  |  | LFSX | LFDX | LFQX |  |  |  |  |  |
| Ax | SB | SC | SH | SW |  |  |  |  | SBX | SCX | SHX | SWX |  |  |  |  |
| Bx | SFS | SFD | SFQ |  |  |  |  |  | SFSX | SFDX | SFQX |  |  |  |  |  |
| Cx | FADD | FSUB | FMUL | FDIV |  |  |  |  |  |  |  |  |  |  |  |  |
| Dx |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Ex |  | NOP |  |  |  |  |  |  |  |  | NOP |  |  |  |  |  |
| Fx |  |  |  |  |  |  |  |  |  |  |  | IMM1 | IMM2 |  |  |  |

02 Group Func

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x | NAND | NOR | ENOR |  | ADD | SUB |  | MUL | DIV | MOD |  |  | AND | OR | EOR |  |
| 1x |  |  |  |  | ADDU | SUBU |  | MULU | DIVU | MODU |  |  | MTSEG | MFSEG | MTSPR | MFSPR |
| 2x | SLL | ROL | SRL | ROR | SRA |  |  |  |  |  |  |  |  |  |  |  |
| 3x | SLL # | ROL # | SRL # | ROR # | SRA # |  |  |  |  |  |  |  |  |  |  | RTI |

61 Group Cond

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x | BEQ | BNE |  |  |  |  | BRA | BRN |  |  | BGE | BLT |  |  | BGEU | BLTU |