Computer Architecture and Organisation

Intel 80960

Digital Assignment - 1

Prepared by: Vibhu Kumar Singh

Registration Number: 19BCE0215

Submitted to: Ushus Elizebeth Zachariah

Slot: B2

INDEX

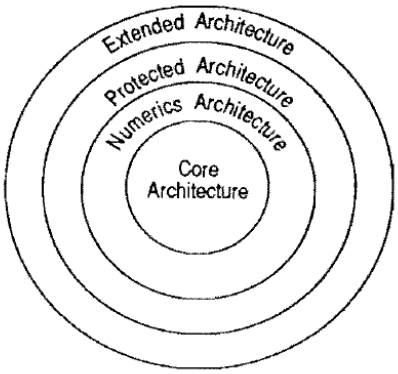
1. ABSTRACT

Intel Corporation's i960 processor own circle of relatives combines architectural functions generally observed in RISC processors with others contained in extra conventional processors. The ensuing own circle of relatives of processors yields excessive performance, but offers a strong and smooth to apply structure version for compiler designers and alertness programmers. This paper discusses many methods wherein software program designers can use the functions of the i960 structure.

2. INTRODUCTION

Intel's i960 structure turned into advanced in the course of the mid-1980's with 3 similarly essential goals: First, it ought to have applicability to a extensive variety of areas, from unmarried chip controllers to large-scale, excessive overall performance devices. Second, it ought to have excessive staying power. And finally, it ought to be optimized for excessive-overall performance implementations, in particular while used with low fee reminiscence systems.

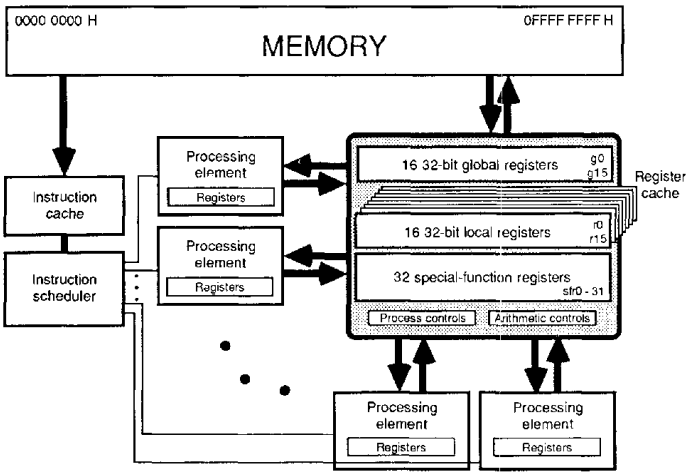
To facilitate the wide range of applications envisioned by the architects of the i960, several architectural layers were defined from the onset(fig 1).



Layers of architecture(Intel 80960)
(fig 1)

Each layer is a complete superset of the layer it encompasses. The first layer, known as the "core", defines the fundamental instructions, registers, and operation of all i960 components. For applications requiring numerics support, a second layer is added which provides full IEEE 754 floating point support, long-double (80-bit) floating point registers, and a robust set of floating point instructions. Applications requiring memory management and hardware support for multitasking (such as those written in ADA) are provided support through implementations of the protected layer of the architecture. The final layer , the extended architecture, provides hardware enforced objects, useful for secure and object oriented language applications.

3. ARCHITECTURE



Architecture Diagram (intel 80960)
(fig 2)

Advances in RISC research occurred simultaneous with much of the i960 architecture definition. Many of the results of this research were applied to the i960 to meet the criteria of a high performance foundation.Some attributes (along with dependence on excessive pace reminiscence subsystems) did now no longer suit nicely to the said dreams of the structure and consequently have been avoided. The resulting architecture has been called everything from a CRISP (complex reduced instruction set processor) to a 'CISCy RISC' to a 'RISCy CISC'.

3.1 Flat Address Space

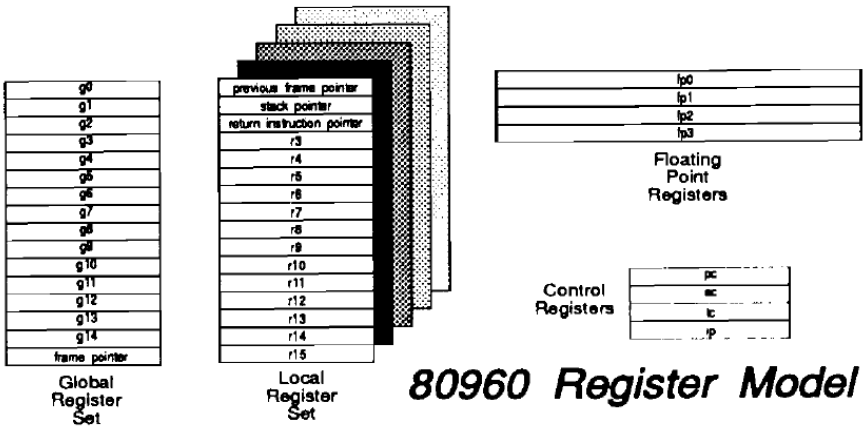
The address space of all i960 processors spans four gigabytes (2^32). Addressing is linear, i.e. there is no segmentation. All input/output is memory mapped. Except for a small boot area and on-chip I/O functions, there are no reserved locations in the address space. Interrupt tables, fault (error condition) tables, and system call tables can be located anywhere in the address space. For processors such as the 80960MC and 80960MX which implement virtual memory, an MMU provides 4K -byte mapping ofvirtual memory to physical memory.

3.2 Data Types

The i960 can access memory as 8,16,32,64,96, or 128-bit quantities. For 8, 16, and 32-bit accesses, integer load/store instructions will sign extend the most significant bit as appropriate. Ordinal instruction are provided for unsigned data. The larger accesses (64,96, and 128-bit) are useful for manipulation of floating point values, data structures, and provide an excellent match to the hardware memory burst capability of the i960. The i960CA exploits this data movement by allowing access to an on-chip SRAM through a 128-bit data bus, allowing up to four registers to be accessed in a single cycle.

3.3 Register Model

The i960 provides thirty two 32-bit general purpose registers that are visible to the programmer at any given time. Of these thirty two registers, four provide linkage information for the i960 call/return mechanism, and twenty eight are freely available to the application. For versions of the i960 that implement the numerics architecture, the processors provides an additional four 80-bit floating point registers. Floating point is not restricted to these registers; any general purpose register (for float operations) or pair of registers (for double operations) can be used.



Register Model
(fig 3)

At first inspection, such a model would appear to create tremendous overhead for allocation of local registers. The i960 avoids this overhead by implementing a special cache (64 registers on the 80960KA/KBIMC, up to 240 on the 80960CA) in which the processor stores previous sets of local registers. The cache behaves like a FIFO; only when it fills is the oldest set of registers flushed to a previously allocated place on the stack. The i960 restores these registers only when the call depth decreases to the level corresponding to the flushed registers.

3.4 Instructions

The i960 is a load/store architecture, which implies that memory operations are disjoint from ALU operations. The i960 exploits this model by allowing concurrent execution of both types of instructions in the machine. Instructions in the i960 typically are three operand, with two sources and one destination. For instructions involving the ALU (called REG instructions), source operands can be any general purpose local or global register, floating point register (for floating point instructions) or literal in the range of 0..31. For memory reference instructions (called MEM instructions), memory addresses can be one of several addressing modes. The i960 was designed to support the simple, high performance addressing modes (direct and register-indirect) normally found in RISC processors, and more complex modes which are not traditionally associated with RISC. The complex addressing modes were added to allow further exploitation of parallelism in the i960, and to improve code density in systems.

Table 1.
80960 instruction summary.

REGISTER OPERATIONS

ARITHMETICS

add[i | o]	Add
addc	Add with Carry
sub[i | o]	Subtract
subc	Subtract with Borrow
mul[i | o]	Multiply
emul	Extended Multiply
div[i | o]	Divide
ediv	Extended Divide
rem[i | o]	Remainder
modi	Modulo Integer
sh[lo | ro | li | ri | di]	Shift

MOVEMENT

mov[l | t | q]	Move registers to registers
lda	Load Address

COMPARISON

cmp[i | o]	Compare
cmpdec[i | o]	Compare and Decrement
cmpinc[i | o]	Compare and Increment
concmp [i | o]	Conditional Compare
test [*]	Test for condition
scanbyte	Scan for matching byte

LOGICAL

and	dst := src1 & src2
andnot	dst := src2 & (~src1)
notand	dst := (~src2) & src1
nand	dst := ~(src2 & src 1)
or	dst := src1 | src2
nor	dst := ~(src2 | src1)
ornot	dst := src2 | (~src1)
notor	dst:= (~src2) | src1
xnor	dst:= (src2 | src1) & ~(src2 & src1)
xor	dst := ~(src2 | src1) | (src2 & src1)
not	dst := ~src
rotate	Rotate Bits

BIT AND BIT FIELD

setbit	Set a Bit
clrbit	Clear a Bit
notbit	Toggle (invert) a Bit
chkbit	Check a Bit and set condition code
alterbit	Change a Bit according to an operand
scanbit	Search src for most significant set bit
spanbit	Search src for most significant cleared bit
extract	Extract specified bit pattern from a word
modify	Modify selected bits in dst with src

CONTROL OPERATIONS

BRANCH

b	Branch (± 2 MByte relative offset)
bx	Branch Extended (32-Bit Indirect Branch)
bal[x]	Branch and Link ("RISC Branch")
b[*]	Branch on Condition
cmpib[*]	Compare Integer and Branch on Condition
cmpob[*]	Compare Ordinal and Branch on Condition

FAULT

fault[*]	Fault on Condition
syncf	Synchronize Faults

PROCEDURE

call	Procedure Call (± 2 MByte relative offset)
callx	Call Extended (32-Bit Indirect Call)
calls	System Procedure Call
ret	Return

ENVIRONMENT

modpc	Modify Process Controls
modac	Modify Arithmetic Controls
modtc	Modify Trace Controls
flushregs	Flush Local Register Cache to Memory

DEBUG

mark	Conditionally generate Trace Fault
fmark	Unconditionally generate Trace Fault

MEMORY OPERATIONS

LOAD/STORE

ld [b | s | l | t | q]	Load
st [b | s | l | t | q]	Store

READ/MODIFY/WRIITE

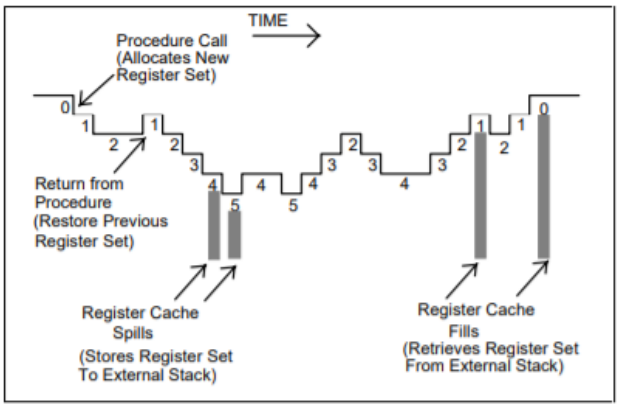
atadd	Atomic Add (Locked RMW Cycles)
atmod	Atomic Modify (Locked RMW Cycles)

A third class of instructions, called CTRL (for control), provide all flow control and call/return. CTRL instructions also may execute independently and concurrently with other instructions in the processor. Unlike other RISC architectures, the i960 does not utilize delay slots after branches. Because the i960 architecture supports superscalar implementations (and hence the ability for several instructions to be in the same pipeline slot), the number of delay slots could not easily be fixed across all parts. The architects chose an alternative method, called branch prediction. Branch prediction takes the form of a bit in the CTRL opcode which provides the processor a hint of the usual path through the branch (i.e. taken or not taken). The control logic of the processor loads the pipeline from this path, and if the hint was correct, the processor incurs no penalty. If the hint was wrong, a pipeline stall occurs.

4. CACHE DESIGN

4.1 Fast call/return mechanism

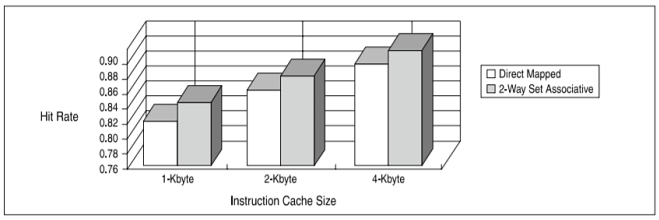
Software performance is crucial in an embedded microprocessor environment. The i960 architecture features a fast call-and-return mechanism to support the modular software base found in today's applications. The local register cache provides storage for the local registers available to software applications. An executing program always has access to sixteen 32-bit local registers and sixteen 32-bit global registers. Executing a procedure call or return (ret) instruction preserves the contents of the global registers across procedure boundaries. The core allocates a unique set of local registers for each new procedure and deallocates them on the return from that procedure. Figure 5 illustrates the available registers and the register cache. Execution of a single call instruction saves the local registers and allocates a new stack frame for the new procedure. The return (ret) instruction deallocates the current local register set and restores the previous state of the machine. The core allocates and deallocates each of these register sets in the local register cache embedded within the processor core. This mechanism provides for fast movement of registers to and from the internal local register cache over a 128-bit wide bus. This feature allows the core to save or restore all sixteen local registers in four clocks.



Register Cache Example
(fig 4)

The i960 RP processor provides enough storage in the local register cache to maintain eight nested calls. When the call mechanism needs additional register sets, the processor will automatically spill (or write to external memory) the oldest register set in the local register cache. This provides a fresh set of local registers for the currently called procedure. Execution of a return instruction will automatically fill (or read from external memory) the current register set with the previously spilled registers. In addition, the register architecture allows the application to reserve register sets for high priority interrupts. Figure 6 shows the register cache behavior over time with four register sets reserved for high priority interrupts.

4.2 Set-Associative Design



Instruction Cache hit rate Comparison
(fig 5)

The i960 RP processor contains three independent caches; instruction cache, data cache, and local register cache. In addition to the on-chip caches, there is a zero wait-state on-chip data RAM. Cache organization refers to the method of associating locations in main memory with locations in the cache. Most embedded RISC processors have direct-mapped instruction caches. In a direct mapped cache, only one specific location in the cache translates into a specific location in main memory. In a two-way set-associative cache, one of the two different locations in the internal cache translates to the specific location in main memory. Academic studies show that a two-way set-associative cache is typically as efficient as a direct mapped cache of twice the size2. Figure 5 shows the cache hit rates for both types of cache organization.

5. PIPE-LINING STAGES

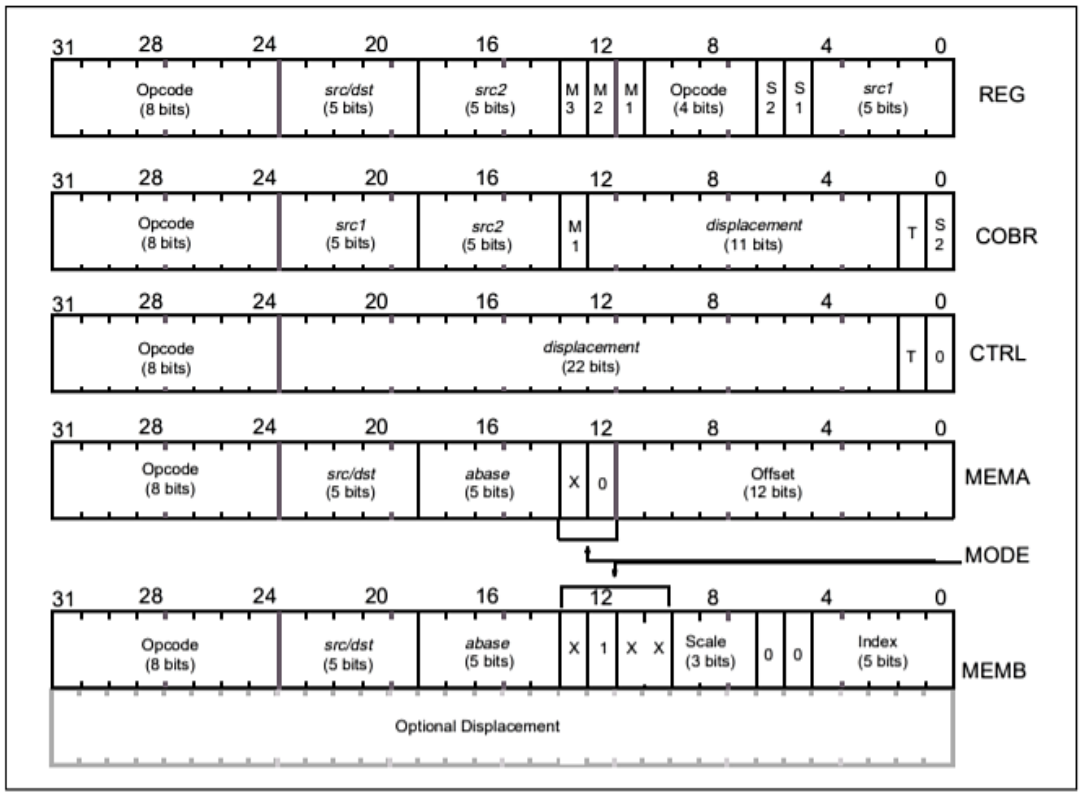
A 32-bit high performance bus controller interfaces the 80960Hx core to the external memory and peripherals. The Bus Control Unit features a maximum transfer rate of 160 Mbytes per second (at a 40 MHz external bus clock frequency). A key advantage of this design is its versatility. The user may independently program the physical and logical attributes of system memory. Physical attributes include wait state profile, bus width, and parity. Logical attributes include cache ability and Big or Little Endian byte order. Internally programmable wait states and 16 separately configurable physical memory

regions allow the processor to interface with a variety of memory subsystems with minimum system complexity. To reduce the effect of wait states, the bus design is decoupled from the core. This lets the processor execute instructions while the bus performs memory accesses independently. The Bus Controller's key features include:

- De-multiplexed, Burst Bus to support most efficient DRAM access modes
- Address Pipe-lining to reduce memory cost while maintaining performance
- 32, 16 and 8-bit modes to facilitate I/O interfacing
- Full internal wait state generation to reduce system cost
- Little and Big Endian support
- Unaligned Access support implemented in hardware
- Three-deep request queue to decouple the bus from the core
- Independent physical and logical address space characteristics

6. GENERAL INSTRUCTION FORMAT

The i960 architecture defines four basic instruction encoding formats: REG, COBR, CTRL and MEM (see Figure). Each instruction uses one of these formats, which is defined by the instruction's opcode field. All instructions are one word long and begin on word boundaries. MEM format instructions are encoded in one of two sub-formats: MEMA or MEMB. MEMB supports an optional second word to hold a displacement value. The following sections describe each format's instruction word fields.



General Instruction Format
(fig 6)

7. INTEL 80960 FAMILY

Intel 80960CA 16 MHz

Intel 80960CA 33 MHz

Intel 80960HD 66 MHz

Intel 80960KA 20 MHz

8. REFERENCES AND CITATION

- D. Ryan, "Intel's 80960: An Architecture Optimized for Embedded Control" in IEEE Micro, vol. 8, no. 03, pp. 63-76, 1988.
  doi: 10.1109/40.541
  keywords: {}
  url: https://doi.ieeecomputersociety.org/10.1109/40.541

- i960® Rx I/O Microprocessor Developer's Manual
  url: http://datasheets.chipdb.org/Intel/80960/manuals/27273602.PDF

- i960® RP Processor: A Single-Chip Intelligent I/O Subsystem
  url: http://datasheets.chipdb.org/Intel/80960/OVERVIEW/1960857.PDF

- 80960HA/HD/HT 32-Bit High-Performance Superscalar Processor
  url: https://media.digikey.com/pdf/Data%20Sheets/Intel%20PDFs/80960HA_HD_HT_Sep02.pdf