1. **Performance:**

**Throughput: total work done in a given time**

**Response time: time between start and completion of a task **

**Applications usually have varied demand on throughput and response time**

**To maximize performance, we want to minimize execution time**

**Performance*X***

**= *n***

**Performance*Y***

**If x is n-times faster than y then :**

**Time is the measure of computer performance - The computer that performs the same amount of work in the least time is the fastest.**

**Different time concepts - Clock time / response time / elapsed time  The total time to complete a task, including disk accesses, memory accesses, I/O operations, operating system overhead, etc.**

**CPU time: the time the CPU spends on a task**

**Does not include time spent waiting for I/O or running other programs.**

**User CPU time: CPU time spent in the program **

**System CPU time: CPU time spent in the operating system performing tasks on behalf of the program**

**Clock cycles: how fast the hardware can perform basic functions.**

**Clock period: the time for a complete clock cycle.**

**Clock rate: the inverse of clock period.**

**CPU execution time formula(appendix paper)**

**Average number of clock cycles per instruction (CPI):appendix paper**

**Average clock cycles per instruction (CPI) provides a way of comparing two different implementations of the same instruction set architecture**

**Instruction set architecture: an abstract interface between the hardware and software that encompasses all the information necessary to write a machine language program. Instructions, registers, memory access, I/O, etc**

**Each computer executes the same number (I) of instructions for the program. CPU clock cycles = I x CPI**

**We can determine which computer is faster by CPU performance and execution time for same instructions.**

** General formula appendix paper**

**Amdahl’s law  The performance enhancement possible with a given improvement is limited by the amount that the improved feature is used.**

**Formula on appendix paper**

**Millions instructions per second(MIPS)**

**Faster computer means bigger MIPS.**

**Problems with using MIPS for comparing computers - Different instruction set - MIPS varies between programs on the same computer**

**Formula on the appendix paper**

**MIPS can vary independently from performance.**

1. **Language:**

**To command a computer’s hardware, you must speak its language.**

**• The words of a computer’s language are called instructions.**

**• The collection of words, the vocabulary, is called the instruction set.**

1. **bit architecture (word = 4 bytes)**

**32 bit addressable space**

**MIPS stands for Microprocessors without Interlocked Pipeline Stages**

**MIPS processors are RISC processors - Restricted Instruction Set Computer - Different from Complex Instruction Set Computer (CISC), such as Intel x86**

**Abstraction is a technique for hiding lower level details of hardware and software to provide a simpler higher level view**

**One of the most important abstractions is the interface between the hardware and the lowest level software**

**Instruction set architecture - Includes machine instructions, I/O devices, etc.**

**Abstraction hides complexity but also hides any issues involved in using the hardware**

**Understanding what lies beneath the abstraction allows us to maximise our use of the hardware**

**Operation: an action to be performed by the computer hardware**

**MIPS operands - 32 registers **

**$s0-$s7, $t0-$t9, $zero, $a0-$a3, $v0-$v1, $gp, $fp, $sp, $ra, $at**

**2^30 memory words**

**Note A word = 4 bytes so it is 2^30 instead of 2^32**

**Types MIPS instructions**

**Arithmetic **

**add, subtract, add immediate**

**Data transfer**

**load word, store word, etc.**

**Logic **

**and, or, shift left logical, shift right logical, etc.**

**Conditional branch**

**branch on equal, branch on not equal, etc.**

**Unconditional**

**jump  Jump, jump and link, etc.**

**The number of operands •**

**The natural number of operands for an operation like addition is three**

**Requiring every instruction to have exactly three operands, no more or no less, conforms to the philosophy of keeping the hardware simple**

**Design principle 1: Simplicity favors regularity**

**A processor register is a small amount of storage available as part of a processor -**

**Registers are built in processor’s datapath -**

**Processor registers are normally at the top of the memory hierarchy.**

**Type of registers**

**- General purpose registers (GPRs): store both data and addresses and can be directly accessed in a user program.**

**- Floating point registers (FPRs): store floating point numbers.**

**- Constant registers: hold read only values such as $zero.**

**- Special purpose registers (SPRs): instruction register, status register, cause register, etc.**

**The number of registers**

**See the appendeix papers**

**Design principle 2: Smaller is faster.**

**Memory operands**

**• The processor can keep only a small amount of data in registers.**

**• Large data structure should be kept in memory.**

**Memory can be viewed as a large, single-dimensional array**

**Data transfer instructions: lw and sw**

**These two instruction access the memory address instead of the value.**

|  |  |  |
| --- | --- | --- |
| **Sequential Addresses** | **Byte array** | **Word Array** |
| **Start at** | **0** | **0** |
| **Up the top of memory** | **2^32 =4GB** | **2^30=1GW** |
| **Step by** | **1 byte** | **4 bytes** |

**Memory vs. registers •**

**Memory is much larger - Up to 1GW compared to 32 W (32 registers of 1W) • Registers are much faster**

**- At least 100 times**

**• Using main memory for manipulating operands would be agonizingly slow • So, we:**

**Transfer data from memory into registers when needed**

**Spill registers back if needed**

**Understand use of registers**

**• Compilers try to keep the most frequently used variables in registers and places the rest in memory.**

**• Spilling registers: the process of putting less commonly used variables (or those needed later) into memory.**

**• Data is more useful in registers**

**- Data in register is both faster to access and simpler to use with less energy demand.**

**- Data usually needs to be moved into a register before use.**

**- MIPS data transfer instruction only reads one operand or writes one operand, without operating on it.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Format** | **Opcode** | **Rs** | **Rt** | **Rd** | **Shamt** | **Funct** |
| **R** | **6** | **5** | **5** | **5** | **5** | **6** |
| **I** | **6** | **5** | **5** | **Constant** | **Or** | **Address (16bits)** |
| **J** | **6(26-31)** | **Pseudo--** | **Address** | **0-25** |  |  |

**Format table on appendix paper**