**SLIDE 9 – Linguagens e Compiladores**

* Software é o que faz a diferença no projeto.
* Muitas linguagens existem.
  + Hardware x software x data flow
* Linguagens hibridas.
* Linguagens servem para especificar a aplicação.
  + Linguagens normais descrevem um comportamento.
  + Linguagens de hardware é mais voltada a estrutura.

**ASSEMBLY**

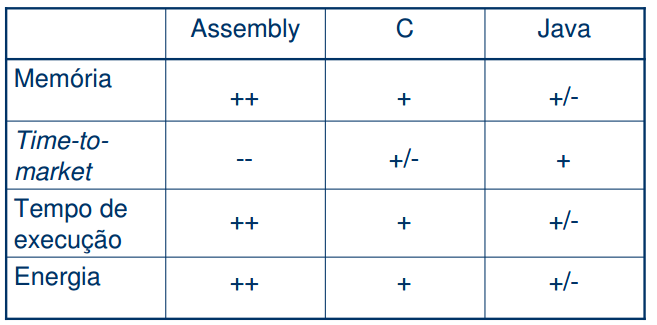
* Usada nos primórdios quando não existiam compiladores, apenas montadores.
* Quando o código era pequeno.
* Linguagem não estruturada, tudo era permitido.
* Poucos programadores
  + Programadores devem ser muito bons para “tirar leite de pedra”
  + Tem de ter profundo conhecimento da arquitetura alvo.
* Aparece nas bibliotecas.
* Tem acesso facilitado ao hardware.
* Tem ótimo desempenho.
* Depuração e reuso são difíceis.

**LINGUAGEM C**

* Assembly estruturado
* Existem muitos programadores, muito utilizado em sistemas embarcados.
* Bom desempenho.
* Fácil acesso ao hardware.
* “Universal”
* Perde muito por usar o compilador, mas perde o mínimo possível
* Depuração e reuso não são muito fáceis.

**LINGUAGEM JAVA**

* Número de programadores elevado.
* Depuração fácil
* Reuso fácil
* Portabilidade: Funciona mais em PC, em SEMB ocorre de faltarem certos métodos e bibliotecas.
  + Pode não implementar tudo
  + Depende do embarcado.
* Pode não ter JVM e gerar bytecode nativo.
* Tem java com simplificação.
* Perde flexibilidade da multiplataforma, mas ganha desempenho compatível com o C
* Pode haver apenas partes da JVM
* JNI promove o acesso ao hardware, interface geralmente com C
* Java é mais rápido no desenvolvimento.
* Tipos de dados mais fortes.
* Não usa ponteiros.
* Usa threads.
* A complexidade da aplicação leva a utilização de linguagens de nível mais alto.
* Garbage collector dar tranquilidade ao programador.
* Não é projetado para tempo real:
  + Falta previsibilidade por conta do escalonamento das threads.
  + Garbage collector entra quando quer e sai quando quer.
  + Existe uma especificação real time para java.



**REQUISISTOS NÃO FUNCIONAIS**

* Coisas que não podemos expressar na linguagem, coisas “por fora” da aplicação.
* O compilador pode alterar o código para obedecer a certos requisitos desses.

**COMPILAÇÃO PARA ENERGIA**

* Escalonamento baseado em energia.
* Seleção de instruções baseado em energia.
* Uso de funções de custo diferente
* Exploração da hierarquia de memória.

**OUTRAS OTIMIZAÇÕES**

* Compilações para DSP, VLIW, multimidia...
* Necessita de um compilador específico.para DSP.
* Gcc é genérico demais torna-o bem reusável, mas não é o mais eficiente.
* Compilador tem que saber do hardware adicional para fazer otimizações.
* Reuso de bibliotecas.
* Quando não tem na biblioteca ou faz com o que tem ou faz as próprias em assembly.
* Lembrar que 70% do código é em assembly e 90% do dsp também.