con [cache_count] [block_size] [policy_num]

Configure System Parameters

- Set Global Configurational Parameters
- Develop Cache List Containing [cache_count] Empty Caches as Doubly-Linked List
- Usually the First Instruction

Parameters

1. **[cache_count]** Number of Cache Layers in this System (Must be At Least 1)
2. **[block_size]** Number of Bytes that Each DataBlock can hold
3. **[policy_num]** Policy Number this System should Implement (Must be Either 0=[Write-Back&Write-Allocate] or 1=[Write-Thru&Non-Write-Allocate])

Requirements

- Must be called ONLY Once.

scd [cache_level] [total_size] [set_assoc]

Set Cache Dimensions

- Set Cache Size and Set Associtivity
- Calculate the Corrct Dimensions for Cache and Field Partitions for Address

Parameters

1. **[cache_level]** The level(index) of cache with lowest being 1
2. **[total_size]** Total Number of Bytes this Cache needs to Store
3. **[set_assoc]** Number of DataBlock for a Each Given Index

Requirements

- Must be called BEFORE **inc**
- Must be called AFTER **con**

scl [cache_level] [latency]

Set Cache Latency

- Set Cache Latency

Parameters

1. **[cache_level]** The level(index) of cache with lowest being 1
2. **[latency]** Number of Clock Cycles to Complete Read for this Cache

Requirements

- Must be called BEFORE **inc**
- Must be called AFTER **con**

sml [latency]

Set Memory Latency

- Set Memory Latency
- Mark Memory Latency as Ready
- *Warning: This Function does NOT Set Memory Latency in Cache Array*

Parameters

1. **[latency]** Number of Clock Cycles to Complete Read from Memory

Requirements

- Must be called BEFORE **inc**

inc [cache_level]

Initialize Cache

- Initialize Cache
- Perform Bound Checks for Cache Level

Parameters

1. **[cache_level]** The level(index) of cache with lowest being 1

Requirements

- Must be called AFTER **con**

tre [address] [arrive_time]

Task Read

- Task Read Address at Time

Parameters

1. **[address]** Raw 32-bit Address to be Read
2. **[arrive_time]** Clock Cycle at when This Specific Task is Scheduled

No Requirements

twr [address] [arrive_time]

Task Write

- Task Write Address at Time

Parameters

1. **[address]** Raw 32-bit Address to be Written
2. **[arrive_time]** Clock Cycle at when This Specific Task is Schedule

No Requirements

pcr [cache_level] [arrive_time]

- Print Cache Hit/Miss Counts and Rates to Report File at Time
- Perform Bound Checks for Cache Level

Parameters

1. **[cache_level]** The level(index) of cache with lowest being 1

No Requirements

pci [cache_level] [arrive_time]

Print Cache Image

- Print the Content of Cache to Report File at Time
- Perform Bound Checks for Cache Level

Parameters

1. **[cache_level]** The level(index) of cache with lowest being 1

No Requirements

ins

- Initialize System
- Run All Tasks in Queue (Read, Write, Print)

No Parameters

Requirements

- Must be called AFTER **inc**
- All **tre**, **twr**, **pcr**,and **pci** need to be schedule BEFORE
- All **tre**, **twr**, **pcr**,and **pci** called AFTER are ignored

hat

Halt Program

- Wait Tasks Queue to be Cleared, then Exit
- Usually the Last Instruction

No Parameters

Requirements

- **Any instructions** called AFTER are ignored

ASSEMBLY CODE GENERATORS

Run the bash file before decoding here:

```
clear all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
int_blockSize = 1;            %Block-size(for determining offset bits)
int_policyNum = 0;            %Policy-num(must be 0 or 1)
intA_setAssoc = [4,8];        %Set-associtivities
intA_latency = [1,50];        %Cache-latency
int_mLatency = 100;            %Memory-latency
intA_cacheSize = 64;        %Total-cache-size(for determining index bits)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tab_processedRW = readtable("processed.txt");
strA_instruction = string.empty();
if(size(tab_processedRW,2)-size(intA_setAssoc,2)~=3||size(intA_setAssoc,2)~=size(intA_latency,2
    error("Dimensions DO NOT MATCH")
end
r = 1;
%%GLOBAL CONFIGURATIONS
strA_instruction(r,1) = "con";
strA_instruction(r,2) = sprintf("$%i",size(intA_setAssoc,2));
strA_instruction(r,3) = sprintf("$%i",round(int_blockSize));
strA_instruction(r,4) = sprintf("$%i",round(int_policyNum));
strA_instruction(r,5) = "//global_configuration";
r=r+1;
%%INDIVIDUAL CACHES
intA_setAssoc = sort(intA_setAssoc);
for i = 1:size(intA_setAssoc,2)%For each cache
    strA_instruction(r,1) = "scd";
    strA_instruction(r,2) = sprintf("$%i",i);
    strA_instruction(r,3) = sprintf("$%i",intA_cacheSize);
    strA_instruction(r,4) = sprintf("$%i",intA_setAssoc(i));
    strA_instruction(r,5) = "//cache_dimension";
    r=r+1;
    strA_instruction(r,1) = "scl";
    strA_instruction(r,2) = sprintf("$%i",i);
    strA_instruction(r,3) = sprintf("$%i",intA_latency(i));
    strA_instruction(r,4) = sprintf(" ");
```

```
    strA_instruction(r,5) = "//cache_latency";
    r=r+1;
    strA_instruction(r,1) = "inc";
    strA_instruction(r,2) = sprintf("$%i",i);
    strA_instruction(r,3) = sprintf(" ");
    strA_instruction(r,4) = sprintf(" ");
    strA_instruction(r,5) = "//initialize_cache";
    r=r+1;
end
%MEMORY LATENCY
strA_instruction(r,1) = "sml";
strA_instruction(r,2) = sprintf("$%i",int_mLatency);
strA_instruction(r,3) = sprintf(" ");
strA_instruction(r,4) = sprintf(" ");
strA_instruction(r,5) = "//memory_latency";
r=r+1;
%%PARSE INSTRUCTION
int_offsetBits = round(log2(int_blockSize));
intA_indexBits = round(log2(intA_cacheSize./intA_setAssoc./int_blockSize));
for i = 1:size(tab_processedRW,1)%For each instruction
    str_comment = "//"+convertCharsToStrings(tab_processedRW{i,1}{1,1});
    str_comment = str_comment+sprintf("[T@%i]",tab_processedRW{i,2});
    if(tab_processedRW{i,1}{1,1}=='r')
        strA_instruction(r,1) = "tre";
    elseif(tab_processedRW{i,1}{1,1}=='w')
        strA_instruction(r,1) = "twr";
    else
        error("ERR Not r or w");
    end
    int_rawAddress = tab_processedRW{i,2};
    for j = size(intA_indexBits,2):-1:1
        int_rawAddress = bitsll(int_rawAddress,intA_indexBits(j))+tab_processedRW{i,2+j};
        str_comment = str_comment+sprintf("[L%i@%i]",j,tab_processedRW{i,2+j});
    end
    int_rawAddress = bitsll(int_rawAddress,int_offsetBits);
    str_comment = str_comment+sprintf("[x]");
    strA_instruction(r,2) = sprintf("$%i",int_rawAddress);
    strA_instruction(r,3) = sprintf("$%i",tab_processedRW{i,size(tab_processedRW,2)});
    strA_instruction(r,4) = sprintf(" ");
    strA_instruction(r,5) = str_comment;
    r=r+1;
    %PARSE ARGUMENTS
end
strA_instruction(r,1) = sprintf("ins");
strA_instruction(r,2) = sprintf("");
strA_instruction(r,3) = sprintf("");
strA_instruction(r,4) = sprintf("");
strA_instruction(r,5) = "//initialize_system";
r=r+1;
strA_instruction(r,1) = sprintf("hat");
```

```
strA_instruction(r,2) = sprintf(" ");
strA_instruction(r,3) = sprintf("");
strA_instruction(r,4) = sprintf("");
strA_instruction(r,5) = "//halt_program";
r=r+1;
clear i;
writematrix(strA_instruction,'instruction.txt','Delimiter','tab');
clear all;
```