**MSE-ASSIGNMENT 3**

1) Explore the DVM instructions and prepare a summary of the same atleast for 5 instructions in a detailed format

i) instruction name:

ii) syntax

iii)example

Android is one of the most popular operating systems (OS) for mobile touchscreen devices, including smart-phones and tablet computers. Dalvik is a process virtual machine (VM) that provides an abstraction layer over the Android OS,and runs the Java-based Android applications.

The normal development environment restricts the Dalvik VM to run on top of Android,and requires an updated Android image to be built and installed on the target device after any change to the Dalvik code. This update-build-install process un-necessarily slows down any Dalvik VM exploration.

However, the Dalvik VM is not distributed as a stand-alone component, and can only execute within the context provided by the Android OS. Likewise, the An-droid open-source community provides no instructions for only building/installing the Dalvik VM sub-component on the host machine.

**Instructins:**

| **Mnemonic** | **Bit Sizes** | **Meaning** |
| --- | --- | --- |
| b | 8 | immediate signed **b**yte |
| c | 16, 32 | **c**onstant pool index |
| f | 16 | inter**f**ace constants (only used in statically linked formats) |
| h | 16 | immediate signed **h**at (high-order bits of a 32- or 64-bit value; low-order bits are all 0) |
| i | 32 | immediate signed **i**nt, or 32-bit float |
| l | 64 | immediate signed **l**ong, or 64-bit double |
| m | 16 | **m**ethod constants (only used in statically linked formats) |
| n | 4 | immediate signed **n**ibble |
| s | 16 | immediate signed **s**hort |
| t | 8, 16, 32 | branch **t**arget |
| x | 0 | no additional data |

**Instruction Set:**

| **Op & Format** | **Mnemonic / Syntax** | **Arguments** | **Description** |
| --- | --- | --- | --- |
| 00 10x | Nop |  | Waste cycles. |
| 01 12x | move vA, vB | A: destination register (4 bits) B: source register (4 bits) | Move the contents of one non-object register to another. |
| 02 22x | move/from16 vAA, vBBBB | A: destination register (8 bits) B: source register (16 bits) | Move the contents of one non-object register to another. |
| 03 32x | move/16 vAAAA, vBBBB | A: destination register (16 bits) B: source register (16 bits) | Move the contents of one non-object register to another. |
| 04 12x | move-wide vA, vB | A: destination register pair (4 bits) B: source register pair (4 bits) | Move the contents of one register-pair to another.  **Note:** It is legal to move from v*N* to either v*N-1* or v*N+1*, so implementations must arrange for both halves of a register pair to be read before anything is written. |
| 05 22x | move-wide/from16 vAA, vBBBB | A: destination register pair (8 bits) B: source register pair (16 bits) | Move the contents of one register-pair to another.  **Note:** Implementation considerations are the same as move-wide, above. |

**Syntax:**

The third column of the format table indicates the human-oriented syntax for instructions which use the indicated format. Each instruction starts with the named opcode and is optionally followed by one or more arguments, themselves separated with commas.

* Wherever an argument refers to a field from the first column, the letter for that field is indicated in the syntax, repeated once for each four bits of the field. For example, an eight-bit field labeled "BB" in the first column would also be labeled "BB" in the syntax column.
* Arguments which name a register have the form "vX". The prefix "v" was chosen instead of the more common "r" exactly to avoid conflicting with (non-virtual) architectures on which a Dalvik virtual machine might be implemented which themselves use the prefix "r" for their registers. That is, this decision makes it possible to talk about both virtual and real registers together without the need for circumlocution.
* Arguments which indicate a literal value have the form "#+X". Some formats indicate literals that only have non-zero bits in their high-order bits; for these, the zeroes are represented explicitly in the syntax, even though they do not appear in the bitwise representation.

**Example:**

* Arguments which indicate a relative instruction address offset have the form "+X".
* Arguments which indicate a literal constant pool index have the form "kind@X", where "kind" indicates which constant pool is being referred to. Each opcode that uses such a format explicitly allows only one kind of constant; see the opcode reference to figure out the correspondence. The four kinds of constant pool are "string" (string pool index), "type" (type pool index), "field" (field pool index), and "meth" (method pool index).

Similar to the representation of constant pool indices, there are also suggested (optional) forms that indicate prelinked offsets or indices. These prelinked values include "vtaboff" (vtable offset), "fieldoff" (field offset), and "iface" (interface pool index).

In the cases where a format value isn't explictly part of the syntax but instead picks a variant, each variant is listed with the prefix "[X=N]" (e.g., "[B=2]") to indicate the correspondence.

**2)**Differentiate between mobile and cloud computing

**Mobile computing**is taking a physical device with you. This could be a laptop or a mobile phone or some device which enables you to [telework](http://en.wikipedia.org/wiki/Telecommuting) working wherever you go because of the small size of the device you’re using.

Mobile apps may use the cloud for both app development as well as hosting. A number of unique characteristics of hosted apps make the mobile cloud different from regular cloud computing.

**Cloud computing:**

The cloud has historically been used as a metaphor for the Internet and is commonly used in network diagrams to represent connections between entities, connected through the Internet

**Main Key differneces:**

* **Total cloud dependency -**When using mobile cloud computing, apps may rely on the cloud for everything, especially when you are trying to develop the same app to run on multiple platforms at the same time using a browser interface. An example of this is an app that runs on Apple [iOS](http://searchconsumerization.techtarget.com/definition/iOS), [Android](http://searchenterpriselinux.techtarget.com/definition/Android), BlackBerry and Microsoft Windows Phone operating systems. Because of the differences between these platforms, developers may rely on the [mobile cloud](http://searchconsumerization.techtarget.com/guides/Consumerization-and-the-cloud-How-mobile-cloud-apps-are-changing-IT) to perform all of the computing and storage to avoid multiple development and maintenance efforts with individual native apps.
* **Mobile cloud computing needs to be communication fault tolerant** – Mobile connections can become weaker and may even disconnect while an application is being used. The cloud may need to be capable of monitoring the connection strength and needs to be fault tolerant of these possible communications disconnects.
* **Distance matters in mobile cloud computing** – Mobile applications when using the mobile cloud may be sensitive to network latencies caused by distance from the server much more than regular [cloud computing](http://searchcloudcomputing.techtarget.com/definition/cloud-computing). The mobile end user experience may suffer if these latencies are too long.

**Conclusion:**

Mobile apps demand a lot more from the mobile cloud than regular cloud computing. Most of these differences are due to the limited energy availability, network latencies and unreliable connectivity in mobile devices. Fortunately, recognizing these differences and adding additional capabilities to the mobile cloud to address them, you can deliver as good an end user experience on mobiles, as you can on desktops and laptops.

3) Give an example of an application simulating an environment of  context aware computing and justify.

Context-aware mobile agent  are a best suited host implementing any context-aware applications.

Modern integrated voice and data communications equips the hospital staff with smart phones to communicate vocally with each other, but preferably to look up the next task to be executed and to capture the next report to be noted.

However, all attempts to support staff with such approaches are hampered till failure of acceptance with the need to look up upon a new event for patient identities, order lists and work schedules. Hence a well suited solution has to get rid of such manual interaction with a tiny screen and therefore serves the user with

* automated identifying actual patient and local environment upon approach,
* automated recording the events with coming to and leaving off the actual patient,
* automated presentation of the orders or service due on the current location and with
* supported documenting the required information keying in a minimum of data into prepared form entries.

Basically such contextually well formed approach requires scheduled workflows, as all necessary preparation must refer to given orders and set schedules. Working free hand or ex tempore does not provide such qualities

Additionally, none of the RFID, WLAN or RTLS locating solutions advertising for *most precise* locating serve the required quality, as determining a location in conventional attitude looking for absolute coordinates fails either technically or economically.

However, the coincidence of personnel with patient is easily detected and delivers all options of reliable automating at reasonable effort. An escape is fairly available with the method of unilateration or fuzzy locating, where no absolute coordinates are required, but just the radial distance between personnel smart phone and patient wrist band.Such distance estimating is entirely sufficient to detect the context of servicing the patient, as no service is ever applied to patients only at larger distance. Even in nuclear medicine or with radiology diagnosis, the haptic contact of staff with patient comes before staff retreats while the patient gets exposed to isotopes or X-rays.